GUARD AGAINST CRIB DEATH WITH YOUR MICRO

THE WONDERFUL DREAMS OF DR. K

THE COMPULSIVE PROGRAMMER by Joseph Weizenbaum

SCOTT JOPLIN ON YOUR SCI-FI HI-FI

ROM

COMPUTER APPLICATIONS FOR LIVING

KILOBYTE CARD
Memory for
Pennies

Volume I, Number 4
October 1977/$2.00

8080
Nan Ardolino

SWTPC announces first dual minifloppy kit under $1,000

MINIFLOPPY DISK SYSTEM
CT-64 TERMINAL SYSTEM
6800 COMPUTER SYSTEM

Now SWTPC offers complete best-buy computer system with $995 dual minifloppy, $500 video terminal/monitor, $395 4K computer.

MINIFLOPPY DISK SYSTEM

$995 MF-68 Dual Minifloppy
You need dual drives to get full benefits from a minifloppy. So we waited to offer a floppy until we could give you a dependable dual system at the right price.

The MF-68 is a complete top-quality minifloppy for your SWTPC Computer. The kit has controller, chassis, cover, power supply, cables, assembly instructions, two highly reliable Shugart drives, and a diskette with the Floppy Disk Operating System (FDOS) and disk BASIC. (A floppy is no better than its operating system, and the MF-68 has one of the best available.) An optional $850 MF-6X kit expands the system to four drives.

$500 Terminal/Monitor
The CT-64 terminal kit offers these premium features: 64-character lines, upper/lower case letters, switchable control character printing, word highlighting, full cursor control, 110-1200 Baud serial interface, and many others. Separately the CT-64 is $325, the 12 MHz CT-VM monitor $175.

6800 COMPUTER SYSTEM

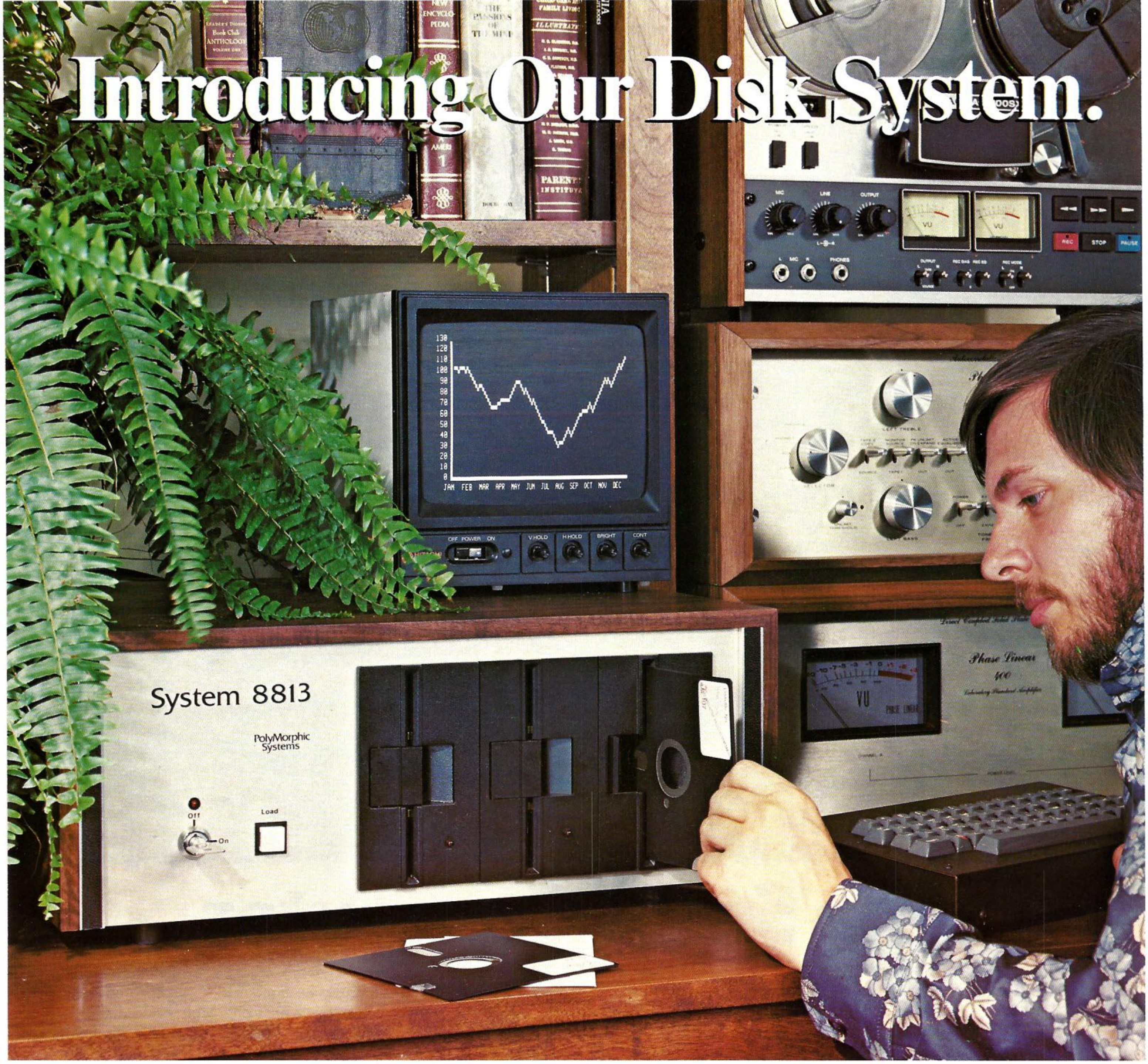
$395 4K 6800 Computer
The SWTPC 6800 comes complete with 4K memory, serial interface, power supply, chassis, famous Motorola MIKBUG® mini-operating system in read-only memory (ROM), and the most complete documentation with any computer kit. Our growing software library includes 4K and 8K BASIC (cassettes $4.95 and $9.95; paper tape $10.00 and $20.00). Extra memory, $100/4K or $250/8K.

Other SWTPC peripherals include $250 PR-40 Alphanumeric Line Printer (40 characters/line, 5 x 7 dot matrix, 75 line/minute speed, compatible with our 6800 computer and MITS/IMSAI); $79.50 AC-30 Cassette Interface System (writes/reads Kansas City standard tapes, controls two recorders, usable with other computers); and other peripherals now and to come.

Enclosed is:

______ $1,990 for the full system shown above (MF-68 Minifloppy, CT-64 Terminal with CT-VM Monitor).
______ $995 for the Dual Minifloppy
______ $325 for the CT-64 Terminal
______ $175 for the CT-VM Monitor
______ $395 for the 4K 6800 Computer
______ $250 for the PR-40 Line Printer
______ $79.50 for AC-30 Cassette Inferface
______ Additional 4K memory boards at $100
______ Additional 8K memory boards at $250
______ Or BAC # ______ Exp. Date ______
______ Or MC # ______ Exp. Date ______
Name ______ Address ______
City ______ State ______ Zip ______

SWTPC
Southwest Technical Products Corp.
219 W. Rhapsody, San Antonio, Texas 78216
London: Southwest Technical Products Co., Ltd.
Tokyo: Southwest Technical Products Corp./Japan


Introducing Our Disk System.
System 8813
PolyMorphic Systems
Off
On
Load
130
120
110
100
90
80
70
60
50
40
30
20
10
0
JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC
OFF POWER ON
V.HOLD
H.HOLD
BRIGHT
CONT

It's like adding a room to your brain!

The System 8813 from PolyMorphic Systems is a complete, powerful problem solver in a single walnut cabinet. This machine allows you to perform complex financial, engineering, and scientific models in the comfort of your office or den.

The high speed video display presents your results in text, tables and graphics. The detachable typewriter-like keyboard permits relaxed program entry and operation. Convenient mini-discs store programs and data for compact filing, secure storage, and fast access. Our disc BASIC programming language is simple enough for the computer newcomer, yet powerful enough to amaze the most advanced users.

The whole family can immediately use and enjoy pre-programmed applications and educational packages. Let the System 8813 become your trusted business, profession, and personal tool. The PolyMorphic Disc System is a completely assembled and tested unit with brushed aluminum front panel, walnut cover, detachable keyboard, video monitor, 16K RAM. Includes system software and fully extended BASIC on disc.

System with 1 disc drive —	$3250
System with 2 disc drives —	3840
System with 3 disc drives —	4430

Delivery 60 days ARO. Upgrade packages for POLY 88 owners will also be available. Prices and specifications subject to change without notice.

PolyMorphic Systems
(805) 967-0468
460 Ward Drive, Santa Barbara, CA 93111

Sol Terminal Computer
Processor Technology

The Small Computer

Twenty-five years ago a computer as powerful as the new Processor Technology SOL-20/8 priced out at a cool million.

Now for only $1350 in kit form or $1850 fully assembled and tested you can have your own small computer with perhaps even more power. It comes in a package about the size of a typewriter. And there's nothing like it on the market today. Not from IBM, Burroughs, DEC, HP or anybody else!

It fills a new role

If you're an engineer, scientist or businessman, the Sol-20 can help you solve many or all of your design problems, help you quantify research, and handle the books too. For not much more than the price of a good calculator, you can have high level computer power.

Use it in the office, lab, plant or home

Sol-20 is a smart terminal for distributed processing. Sol-20 is a stand alone computer for data collection, handling and analysis. Sol-20 is a text editor. In fact, Sol-20 is the key element of a full fledged computer system including hardware, software and peripheral gear. It's a computer system with a keyboard, extra memory, I/O interfaces, factory backup, service notes, users group.

It's a computer you can take home after hours to play or create sophisticated games, do your personal books and taxes, and a whole host of other tasks.

Those of you who are familiar with small computers will recognize what an advance the Sol-20 is.

Sol-20 offers all these features as standard:

8080 microprocessor — 1024 character video display circuitry — control PROM memory — 9216 words of static low-power RAM — 2048 words of preprogrammed PROM — built-in cassette interface capable of controlling two recorders at 1200 bits per second — both parallel and serial standardized interface connectors — a complete power supply including ultra quiet fan — a beautiful case with solid walnut sides — software which includes a preprogrammed PROM personality module and a data cassette with BASIC-5 language plus two sophisticated computer video games — the ability to work with all S-100 bus products.

Full expansion capability

Tailor the Sol-20 system to your applications with our complete line of peripheral products. These include the video monitor, audio cassette and digital tape systems, dual floppy disc system, expansion memories, and interfaces.

Write for our new 22 page catalog. Get all the details.

Processor Technology, Box O, 6200 Hollis St.,
Emeryville, CA 94608. (415) 652-8080.

Processor Technology

Volume I, Number 4

Contents

October 1977

FEATURES

16 From Bombs to ROMs
—Lavinia Dimond
Designing systems in the wilderness.

17-24 missing

20 Guard against Crib Death with Your Micro
—Jon Glick
A microbased childsaver for your home.

27 Home Computers: The Products America May Never Know It Needs
—Martin Himmelfarb
Mass marketing may not work unless....

32 Putting Two & Two Together
—Tom Pittman
Binary arithmetic explained for the beginner.

42 The Wonderful Dreams of Dr. K
—Hesh Wiener
Microprocessor aid for the deaf-blind.

49 The Kilobyte Card: Memory for Pennies
—Thorn Veblen
The handiest, hardiest, cheapest memory system ever.

55 The Unlikely Birth of A Computer Artist
—Richard Helmick
Mix computers with silkscreen and look what you get.

60 Scott Joplin on Your Sci-Fi Hi-Fi
—Dorothy Siegel
Programs from the master riffer for your micro, complete with a mini music guide.

66 Building a Basic Music Board
—Eben F. Ostby
Make me some music, Maestro micro.

68 The Compulsive Programmer
—Joseph Weizenbaum
Dedicated, devoted, and in search of power, the programmer loses control of both the machine and himself.

78-84 missing

80 The Very Best Defense
(A short story)
—Laurence M. Janifer
How do you convince a pacifist computer to KO the nasties?

84 Chart Up and Flow Right
—Eben F. Ostby
Flowcharting demystified for the first timer.

89 Small Business Payroll Program Follow-Up
—Robert G. Forbes
Further details on last month's program listing.

COLUMNS

10 Missionary Position by Theodor Nelson
Access structure and life.

14 The Human Factor by Andrew Singer
It can't be done.

96 Legal ROMifications by Peter Feilbogen
You, Inc?

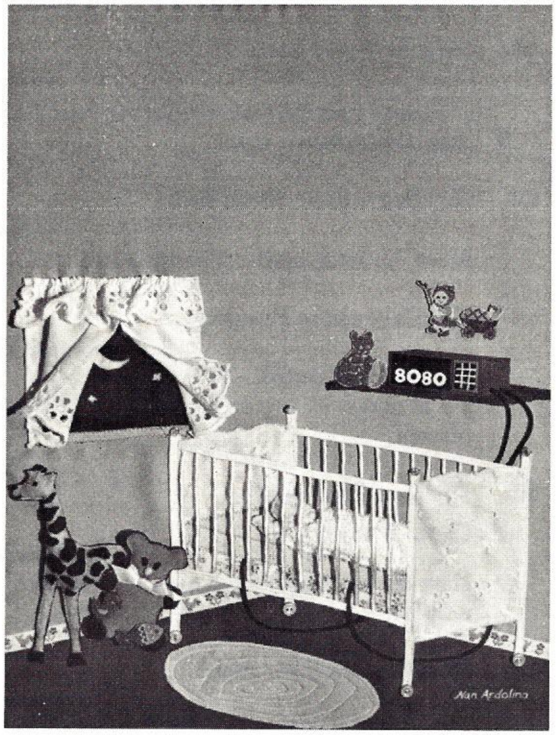
98 FutuROMa by Bill Etra
Playing the percentages.

DEPARTMENTS

4 **On The Bus**
8 **Reader Interrupt**
12 **Eve 'n' Parity**
49 **Run On Micros**
50 **Centerfold**
98 **Babbage and Lovelace**
100 **PROMpuzzle**

ROM is published monthly by ROM Publications Corporation, Route 97, Hampton, CT 06247 (Tel. 203-455-9591). Domestic subscriptions are $15 for one year, $28 for two years, and $39 for three years. Canada and Mexico $17 for one year, $30 for two years, and $41 for three years. For European and South American subscriptions, please add $12 per year additional postage. For all other continents, please add $24 per year additional postage. *Copyright © 1977 by ROM Publications Corporation. All rights reserved.* Reproduction in any form or by any means of any portion of this periodical without the written consent of the publisher is strictly prohibited. The following trademarks are pending: Babbage and Lovelace, Eve 'n' Parity, floppyROM, futuROMa, The Human Factor, Legal ROMifications, Missionary Position, The Noisy Channel, On the Bus, PROMpuzzle, Reader Interrupt, ROMdisk, ROMshelf, ROMtutorial, and Run On Micros. Opinions expressed by authors are not necessarily those of *ROM* magazine, its editors, staff, or employees. No warranties or guarantees explicit or implied are intended by publication. Application to mail at second-class postage rate pending at Hampton, CT 06247. Membership in Audit Bureau of Circulation pending.

On the Bus



Nan Ardolino, a recent graduate of Parsons School of Design, is a free lance illustrator. Ardolino's unique use of material has created a new textural dimension. The fabric construction is appealing to the sense of touch as well as vision, simulating an indefinable third dimension when reproduced in printed form as on this month's cover of *ROM*. Black and white work, watercolors, and pastels, are also media of which she is particularly fond.

Hooked on crossword puzzles at an early age, **Daniel Alber** now constructs as well as solves them. Part of the Brownstone Renovation generation in New York, when he's not constructing puzzles for the likes of *Field and Stream, The New York Times,* and *ROM,* he's reconstructing olden golden rooms in his Brooklyn based house.

Lavinia Dimond is a 32-year-old former Antioch College student, mother, teacher, and writer, not always in that order. Based in the San Francisco Bay area, she teaches first grade in a fifty-year-old alternative school. Her prime goal in life, aside from opening the San Francisco Peninsula's most exciting bistro, is to compile the masses of bizarre, diverse experiences of her life into the *definitive* American novel. Her connection with computers? She happened to fall in love with a mad scientist.

Bill Etra is a West Coast based computer design consultant. He is co-inventor of the Rutt/Etra Video Synthesizer—the first portable voltage control analog video synthesizer, as well as the Videolab. His main interest is videographics, and many of his works have appeared as cover illustrations on various periodicals and books including *Computers in Society* and *Broadcast Management and Engineering*. His current research centers on "The Computer as a Compositional Tool for Video."

Peter Feilbogen attended the Rutgers School of Business Administration and Brooklyn Law School. In addition to being an attorney, he is also a Certified Public Accountant. He has been a sole practitioner on Long Island for approximately ten years, and treasurer of Data Information Services, Inc. An avid tennis player, he is currently trying to improve his game with the aid of a computer.

While **Robert G. Forbes** was taking a psychology course in college, he was taught how to do statistical analysis with a computer. Psychology fell by the wayside, as he switched his major to computer science. Currently a programmer with the May Company, he has written a number of business-oriented programs, both large and small. On the personal computing level, he's waiting for the minis to open up.

Jon Glick has been a scientist ever since his childhood tinkering in his parents' New Jersey basement. His early concoctions (from high altitude rockets to microphotographic equipment for analyzing chemicals) led to prizes in high school but also an arrest for blowing up a neighbor's gazebo. A subsequent checkered academic career eventually put him in the main stream, which he left a few years ago to found Modal Systems, his own business, in Redwood City, California. His main concern since has been building research equipment for infant and child care.

Richard Helmick is Associate Professor of Housing and Interior Design at the University of Missouri in Columbia, Missouri. After earning Bachelor's and Master's degrees in Fine Arts at Ohio University, he taught for a while at Christian College in Columbia. Originally a sculpture major, Helmick has turned to different forms in recent years. He has works on exhibit in the permanent collection of The Missouri State Historical Society, at Indiana State University, and at William Woods College. He has also exhibited in juried and invitational shows in the United States and Canada.

Martin Himmelfarb has been editor of *Digital Design* magazine and business editor of *Minicomputer News.* An electrical engineer with a master's degree in journalism, Himmelfarb is Lecturer in marketing at Suffolk University in Boston. He has built a home computer, but finding no real personal applications, gave it away.

Laurence M. Janifer is married to sci-fi writer Beverly Goldberg, who had the original notion for "The Very Best Defense." Gerald Knave, the central character of this month's short story, is one of Mr. Janifer's valued friends. Knave has had a good many odd adventures, and once in a while he tells some to Mr. Janifer. This is one; a longer and funnier one jumped into print in August, as an Ace paperback titled *Survivor,* and a very good one indeed is in the works, also for Ace, under the title of *Knave in Hand.*

Theodor Nelson is the author of the classic *Computer Lib/Dream Machines,* a Whole Earth style catalogue of computer machinations. Presently at the Department of Mathematics of Swarthmore College where he is working on the Hypertext Project, Ted specializes in highly interactive systems for graphics and text. His past experience includes a stint at Dr. Lilly's Dolphin Laboratory and work as a consultant for Bell Lab's ABM system.

Eben F. Ostby first began experimenting with computers while at the Pomfret School, which had a PDP-8. He went on to become the first Computer Science major at Vassar College, from which he recently graduated with honors. Ostby has done extensive work with graphics in APL, and recently programmed what he thinks may be the first cartographic project in APL.

Tom Pittman's dream, ever since high school, was to own a computer. Well, now he does—in fact he owns several. Starting with an Intel prototype 4004 system in 1972, he has branched out, turning his hobby into a business, albeit a tiny one, as the main implementor of Tiny BASIC.

Dorothy Siegel, a graduate of Smith College and Yale University, is interested in both classical music and computer applications. Her collection of musical instruments includes, beside her clarinet, an IMSAI computer with floppy disk, Diablo printer, and Model 6 Music Board—the last manufactured by her company, Newtech Computer Systems, Inc., of Brooklyn, New York.

Andrew Singer has been hooked on computers since he first built one in 1958. A hard/software consultant, he is fluent in thirty computer languages and knows more than enough about twenty species of machine. His work has included the first medical information retrieval system based on ordinary clinical records, and a large and intricate system for interactive selection of data from public opinion polls. He believes that most software is poorly designed and unspeakably rude, and his Ph.D. research is aimed at improving the architecture and human engineering of interactive systems.

Thorn Veblen is a free lance economist specializing in human value analysis in bionics. He claims to have no upper-class pretensions except for a preference for playing croquet on putting-green-smooth lawns.

Joseph Weizenbaum is Professor of Computer Science at the Massachusetts Institute of Technology. He is best known to his colleagues as the composer of SLIP, a list-processing computer language, and for ELIZA, a natural-language processing system. More recently, he has directed his attention to the impact of science and technology—and of the computer in particular—on society.

Hesh Wiener is the editor of *Computer Decisions,* one of the most widely read computer trade publications in the United States. Previously he was on the academic staff of the University of California at Berkeley, where he headed the Computer Education Project at the Lawrence Hall of Science. During that time he spent a year teaching blind children to use computers. He designed and built some of the equipment used by the blind students. ▼

FREE!
with your business card or send $1.00 (refundable on 1st order)

Newman's 68 Page Fact-Filled Mini-micro Computer Catalog

LARGEST CATALOG IN ITS FIELD

Catalog Includes:

- Manufacturer's catalogs.
- News about amazing breakthroughs in the Mini-micro computer field.
- $2.00 Discount Certificate.
- Discounts up to 90% on Used Equipment.

·A Complete Computer System Just $289

COMPUTER DEALERS ASSOCIATION

Everything:

- **Fully Assembled**
- **Fully tested**
- **Fully Warranted**
- **KIM-1** — MOS Technology Computer Module 1K-RAM, audio cassette interface, 15 bidirectional I/O lines, 24-key keyboard, and six-digit LED display.
- **System Power Supply** (5V at 1.2A, 12V at 0.1A), with power line and switch.
- **Software** — System Executive, Sample application programs.
- **Documentation.**
  User hardware & programming manuals, wall size System Schematic

  Programmer's Reference Card

Money-Back Guarantee

Return items undamaged for any reason within 10 days of receipt and get a complete refund.

·Computer Books

An Introduction to Microcomputers, Vol 1 - Basic Concepts (Osborne). A complete book. $7.50

Vol 2 - Some Real Products. Details today's real products. $12.50

Basic BASIC: An Introduction to Computer Programming in BASIC Language (Coan) An excellent introduction to BASIC. $7.95

Advanced BASIC: Applications and Problems (Coan) Advanced techniques and applications. $8.95

Hobby Computers are HERE Green Simplified introductions to various aspects of hobby computing. $4.95

Telephone Accessories You Can Build (Guilder). Remote telephone ringing, speech scrambler for privacy, automatic dialing, etc. $3.95

Bankard/Visa &
Master Charge accepted

Order From:

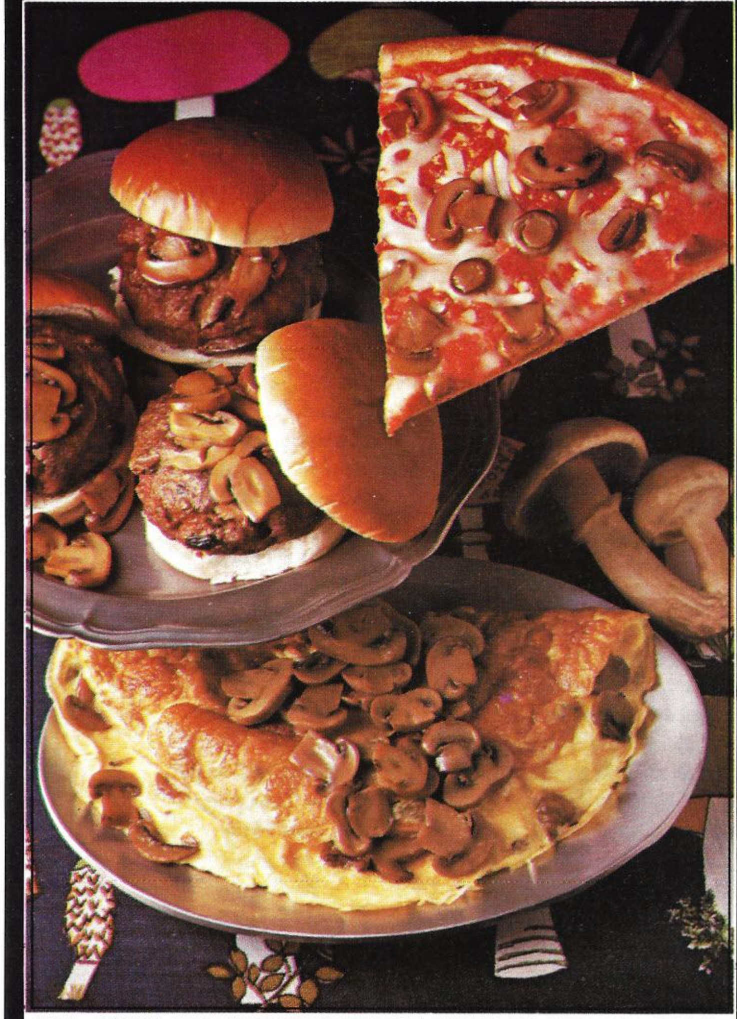
NEWMAN COMPUTER EXCHANGE
1250 N. Main St. Ann Arbor, Mi. 48104
(313) 994-3200 Dept. R

COLLIER IS THE BEST ENGRAVER, PRINTER IN THE COUNTRY.

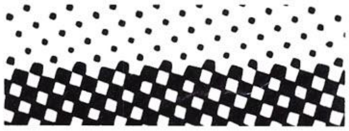
Because. A revolution in engraving has happened. Collier split the dot. And because of the new Laser Beam, Collier is now light years ahead of the rest. The laser does it. Here’s how it works. Technically, the laser beam which is used to control film exposure (thus “Program” the engraving) is split into six sectional beams. Each of these is digitally modulated by computer to transfer half-dots of picture information per scanner revolution. Since two picture-information “bits” are available per dot in circumferential direction, it’s possible to expose a smaller area in the second “orbit”. . . or even completely omit it (thus producing an actual half-dot or even an elliptical mini-dot). If that seems to all sound a lot like gobbledygook, don’t worry about it. The important thing is, it’s here and it does work and it gives your image resolution that you never could get before. We’ll be happy to demonstrate it for you. Even happier to produce your next set of four-color letterpress, offset and gravure engravings. Call Collier Graphic Service Company Inc., 240 West 40th Street, New York, New York 10018/(212) 840-0440.

ATLANTA	BOSTON	DETROIT	HARTFORD	NEW YORK	PHILADELPHIA
(404) 892-2383	(617) 965-5660	(313) 259-2111	(203) 367-0706	(212) 840-0440	(215) 988-0110

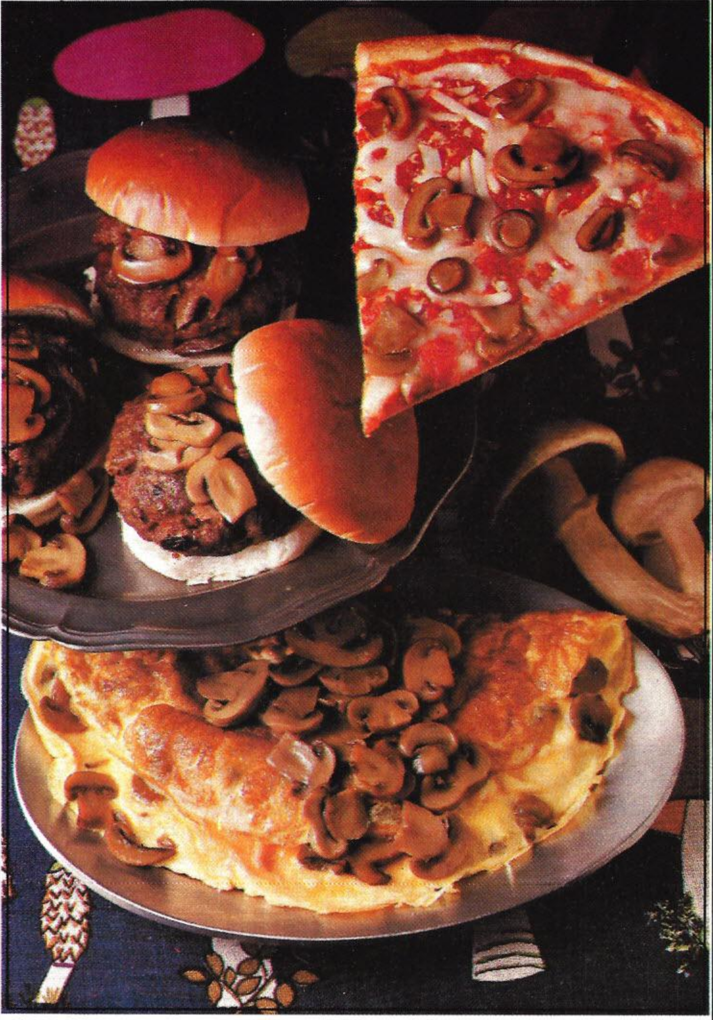
OR CALL TOLL FREE (800) 221-2585/(800) 221-2586



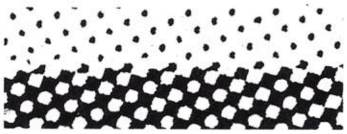
THE ABOVE ENGRAVING WAS MADE WITH THE CONVENTIONAL PLATE MAKING PROCEDURES, AS YOU CAN SEE, IT LACKS COLOR FIDELITY AND SHARPNESS, IF YOU COMPARE IT WITH...



THE CONVENTIONAL ENGRAVING DOT.



THE COLLIER'S LASER BEAM ENGRAVING, AS YOU CAN SEE FROM THE ABOVE, HAS BRILLIANCE, SHARPNESS AND COLOR FIDELITY THAT ONLY COLLIER'S DOT CAN PROVIDE.



THE COLLIER LASER DOT.

THE COLLIER ADVANTAGE. COLLIER GRAPHIC SERVICES COMPANY, INC., 240 WEST 40TH STREET, NEW YORK, NEW YORK 10018

Reader Interrupt

Dear ROM,

When I saw the first issue of *ROM* on sale at one of the local computer stores I thought "Oh no, not *another* one!" I bought a copy anyway and was pleasantly surprised to find that *ROM* isn't just a silicon copy of the other hobby mags but has a personality all its own. I especially liked Gordon Morrison's article and hope that *ROM* has more of the same on tap.

When I got to Andrew Singer's column I immediately recognized the infamous CP-V COPY command from my days as a tech writer at Xerox. No, I didn't write the manual in question but did write some others equally confusing. The problem seems to arise from the fact that too little planning goes into the design of the user/system interface. System programmers are too busy writing code to give much thought to what the system commands and messages should look like. And even if they did think about it, few software designers have had any training in linguistics.

Actually, all software projects *should* begin with a careful design study of the proposed user/system dialogue, including the command language and any messages to the user. The services of a semanticist or human-factors expert at this point would save countless hours of user frustration later, when the software is cast in concrete. But this hardly ever happens, and the result is tragic. The best system in the world is worthless if the user can't communicate with it in a reasonable way.

Jim Day
Granada Hills, California

Dear ROM:

I am enjoying the first issue of *ROM* and am particularly pleased to see your ROMtutorial definitions accompanying each article. We computer types are bad about not defining our terms, assuming that others understand them, and this feature of your journal is most helpful.

One comment needs to be made, I think, about Sandra Faye Carroll's article "A Chip Is Born." She left out what I consider a very interesting part of the process, namely, the production of the silicon rods which are sliced into the "irridescent teal blue disks" she refers to. Float-zone refining is an intriguing technology itself and might well be of interest to your readers. How about an article on it in a future issue?

(Mrs.) Rusty Harris
Atlanta, Georgia

Coming up soon.

ROM

Dear ROM,

I am writing about the SOC. Is this in some way a descendent of McCulloch's work on the reticular formation? I gather it is now an accepted device, at least in some circles—People actually build them and use them for process control! I would be very interested in any references you have available on the construction, theory, or uses of SOCs. My own background is primarily software, with some math and a smattering of hardware.

My tastes in articles run towards "How we did it" rather than "The general theory of. . . ." (unless it really is the general theory of something interesting).

If you have any, I'd also be interested in references on the "soft control material." Sounds like it might go well with a SOC-oriented system.

Jed Harris
New York, New York

SOCs have more or less been standing still, and the best reference on the topic is still the papers, including those from Roger Barron and Lewey O. Gilstrop, from the Bionic Symposium held at Wright Patterson Field at Dayton, Ohio in 1962. The relation of the SOC and reticular formation is actually contrary, rather than complementary, since RF chooses habits, while the SOC establishes no habit. Its beauty lies in being habit-free.

More on SOCs and soft control in a future issue—as soon as Avery Johnson finishes building his new homestead, hopefully in a few months.

ROM

Editor and Publisher
Erik Sandberg-Diment

West Coast Editor
Lee Felsenstein

Associate Editor
Janet C. Robertson

Contributing Editors
Sandra Faye Carroll
Bill Etra
Louise Etra
Ed Hershberger
Avery Johnson
Richard W. Langer
Theodor Nelson
Robert Osband
Eben F. Ostby
Frederik Pohl
Andrew Singer
Alvin Toffler

Crossword Puzzle Editor
Daniel Alber

Art Director
Susan Reid

Contributing Artists
Robert Grossman
Cindy Hain
Luis Jimenez
Korkie
David Macaulay
Rex Ruden
Linda Smythe

Art Assistant
Sue Bass

Staff Photographer
Thomas Hall

Composition
Lynn Archer

Editorial Assistant
Donna Parson

Special Assistants
Jennifer L. Burr
Joanne Zeiger

Counsel
Peter Feilbogen

A Note to Contributors:
ROM is always looking for good computer applications articles from people with up-and-running systems. We also will be glad to consider for possible publication manuscripts, drawings, and photographs on other computer-related subjects. Manuscripts should be typewritten double-spaced, and a stamped self-addressed envelope of the appropriate size should accompany each unsolicited submission. Although we cannot assume responsibility for loss or damage, all material will be treated with care while in our hands. Contributions should be sent to ROM Publications Corporation, Rte. 97, Hampton, CT 06247.

You're curious enough to read ROM now, you'll be curious enough to read ROM every month. So subscribe!

SAVE $ 7.00 over the single-copy price with a one-year subscription

SAVE $20.00 over the single-copy price with a two-year subscription

SAVE $33.00 over the single-copy price with a three-year subscription

SAVE EVEN MORE
Rich Uncle Special:

For the first six months, and the first six months only, ROM is offering genuine inflation-proof lifetime subscriptions at $250.00 each. No matter what happens to the dollar compared to the yen, the mark, the franc, or even the yak, with the Rich Uncle Special you're assured of a lifetime supply of ROMs. At today's rate of inflation, a year's subscription may well be selling for that much in only a decade.

If you wish to be billed, please use either Master Charge or BankAmericard. Direct billing by the publisher is the single largest cause of subscription service problems. With today's postage rates, it is also very expensive for you as well as ROM.

GUARANTEE:
If not satisfied with **ROM** at any time, let us know. We'll cancel your subscription and mail you a full refund on all copies still due you.

ROM
COMPUTER APPLICATIONS FOR LIVING

ROM Publications Corp.
Route 97
Hampton, CT 06247

Name
Address
City
State Zip
☐One year $15 ☐Two years $28 ☐Three years $39
☐Check or money order enclosed. ☐Lifetime $250
☐Master Charge ☐BankAmericard
Exp. date Card#

Please allow 4-6 weeks for delivery.

Missionary Position

ACCESS STRUCTURE AND LIFE

by Theodor Nelson

Starting with a simple technical distinction, which I've tried to generalize a little, I've come up with an interesting model of what we may call "access structure." It seems to have ramifications for a lot of matters: design of systems, storage, objects. And indeed, much of life is concerned with access structure.

This originally came to mind in the early sixties when I was thinking about mass storage. Obviously, there is a big distinction between information that's in core memory, where it can be gotten at instantly, and information that's on disk, requiring a fetch and consequent delay.

As I thought about this general distinction between near things and far things, it seemed to me applicable as well to everyday life. Mine, anyhow. People who don't understand how I look at access structure are dangerous to let into places I live. This form of analysis may or may not be useful in business or industry—they already have some inkling of it—but it may help us think about homes (you know, boxes for living), and the artifacts that people have in homes. I think homes are as a rule unspeakably irrational.

Suppose something is in your hand, a knife or a pencil or a pair of pliers, poised and ready for use. You've got it, it's ready, there is no impediment. Let us define this condition as *zero-order access* to this object.

Next, suppose the object is on the table and you have to pick it up. Nothing is physically in the way, but you do have to take the step. This is *first-order access.* One operation will bring the object to zero-order access; the space between your hand and the object is an impediment of sorts to be overcome by the intervening step.

But suppose now that some blocking object is in the way, perhaps in front of the object you want, perhaps just a sheet of paper covering it. This condition constitutes an impediment holding you back. The blocking object must be pushed aside, or the sheet of paper picked up, to render the object seizable. So now we say the object is at *second-order access,* because an operation stands between you and its first-order access.

Clearly the terminology may be extended indefinitely, to the tenth-and *nth*-order access. When we consider the way the world is laid out, and the different things we want to use, we begin to see compound access structures all over. (I'm not distinguishing here between sizes or shadings of accessibility. An intervening step is one access step, no matter how large or small. Obviously the model can be adjusted where this becomes inappropriate.)

This model is intended, among other things, to clarify and enrich the basic concept of "in the way": something which is "in the way" may be characterized for its exact effects on access structure.

There is a slight terminological problem here. Since this model of access structure is developed by induction on the basis of further and further removes from instant readiness, it makes sense to start at zero. Unfortunately, there is a clash between the phrase "high accessibility" and my term "low-order access"—which mean about the same thing. So access, as defined by access *order,* is the inverse of the more common term "accessibility."

Access may be serial or parallel. Serial or segregated access to a collection of things means we cannot get at them simultaneously: we can only get access to them one at a time but not jointly. Examples are conveyor belts and lazy susans.

Parallel or joint access means we can get at different items from a collection simultaneously and together, as from a bookshelf. In other words, several things are simultaneously at low-order access levels.

Many familiar objects are concerned with access structure. These we may call *access machines.* The most fundamental are counters and shelves.

To understand this, first think of putting things on the lawn or in a field. The ground is a two-dimensional place to put things, the elemental storage space.

If you've ever seen a lawn sale, you know how little it takes to fill up a lawn, especially if lanes are left to walk through.

The first thing we can do is raise the lawn up (except for the access lanes), so we don't have to squat down to pick things up—improving access by one order. Thus we have *counters.*

But more room is needed. The next step, conceptually, is to create further holding-surfaces or different levels. Rather than platform over the whole lawn, of course, we'll keep the access lanes between the holding-surfaces. The result is called *shelving.*

We spoke of the ground as the elemental storage space. Well, shelves permit several layers of ground to put things on, all in the same two-dimensional area. An access step is involved in reaching up or crouching down, but the packing density is much greater.

To help think about other access machines, first imagine yourself sitting on the ground with your things in a circle around you. Then only about ten small objects, laid around you, can be at first-order access. And that's pretty much it. But with pegboards and toolchests and swinging trays, a craftsman such as a cabinet-maker or a dentist can have hundreds of items at first and second-order access.

This is the glory of simple access machines. Access machinery includes shelves, lazy susans, pegboards, grabbing devices. (Drawers are merely enclosed shelves, as are cabinets. The extra access step is justified either for safety or prettiness.)

A word here against deep shelving. Shallow shelves permit first-order access; this is the spicerack principle. Nothing is blocked, you can see it all in parallel. But as soon as you start putting things *behind* each other, access deteriorates in several ways. To get at the things in back you'll have to either reach around the first stuff with diffi-

Illustrated by the author

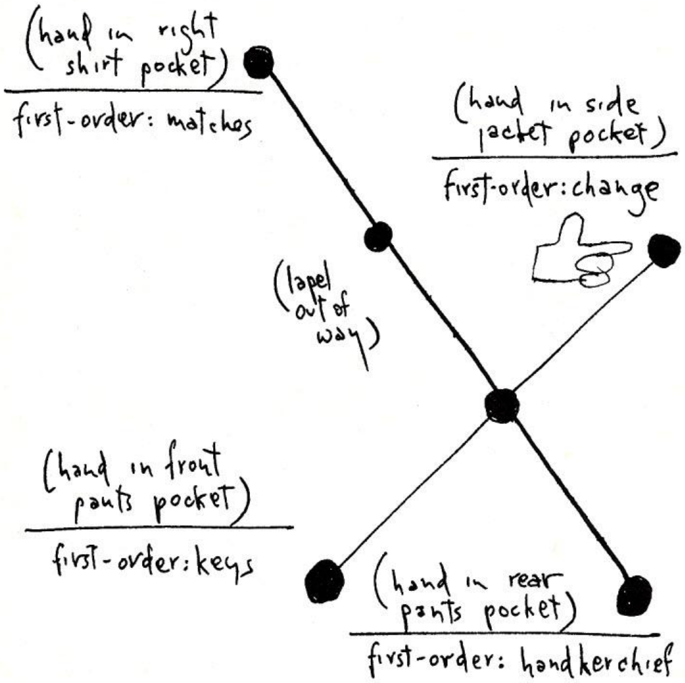
culty, which counts as a lost access step, or remove and buffer the front stuff somewhere else, like the floor, which makes at least two access steps (if you put the stuff back).

Worse, deep shelves and deep closets are not rational storage space, they are places to lose things. More on that later.

Because we do not usually think explicitly about these things, we do not have the access machines we ought to. At the turn of the century you could get modular mobile shelving—the glass-front sectional bookcases—but not now. Overhead hooking arrays, swinging-shelf arrays, all sorts of other access devices ought to be available commercially—but aren't.

Access analysis can be graphed; we can create access graphs of many objects and situations. Typically such a graph consists of points, lines between them, and a Reacher—person, hand or machine—which moves along the graph.

For example, we may graph the access structure of a man's right-hand pockets, and the Reacher, which is his right hand. The result is a star:


(hand in right shirt pocket)
first-order: matches
(hand in side jacket pocket)
first-order: change
(lapel out of way)
(hand in front pants pocket)
first-order: keys
(hand in rear pants pocket)
first-order: handkerchief

If the man's right hand is in the right jacket pocket, his change is at first-order access (since it must still be *grasped*), and the handkerchief in his pants pocket is at third-order access.

If the hand comes out, the change is now at second-order access and so is the handkerchief.

The map delineates states of a Reacher that moves between tasks: now it is closer to one thing, now to another. That is typical access structure, where a step toward one node tends to be a step away from another.

Access machines all have analyzable structures, some graphable. For instance, the access graph of a row of shelves is a two-dimensional grid. The access graph of a lazy susan is a circle which the Reacher touches at one point. A pair of pantographic grabbing tongs lowers the access order of all objects in its reach, since you no longer have to get out of your chair to glom onto them.

All the world's an access structure.

We may regard the world as consisting of process areas, buffers, storage, access lanes, and vistas. (I may have left out something, but never mind.)

Let's run through these.

A process area is where you (or some tool) have access to a thing to work on it.

A buffer is, of course, a place where you put things temporarily. Things do happen in buffer areas, though: dishes dry, dough rises, ideas jell, prisoners become old and discouraged.

Storage is where you put things non-temporarily. Storage is like buffering, but for a longer period. This may be as precise as we can get.

An access lane is a place that is kept clear (of either processes or buffering) in order to permit accesses through it.

A vista is a place that does not hold all the shelving it possibly could, or buffers or workspaces, for aesthetic reasons.

There are other places, but they do not concern us here. Now let's try on some insights:

- A chair is a body buffer; often it grants access to other access areas.
- A table is a counter with leg-space for closer access from a body buffer.
- Stairs and hallways and elevator shafts are access lanes.
- A theater is where people sit in body buffers offering visual access to a process area. (Some people prefer body buffers next to the access lane.)
- A factory is a place where workers have joint access to tools, materials, and products being created.
- A library is a place where books are stored and accessed. Sometimes they have privileged access areas (closed shelves.) There are also process areas where books are read and notes taken.
- An office is a place where papers are stored and accessed. New paper is generated from them at desks, which are, of course, just tables with drawers.

These principles apply as well to the private home. The kitchen is a prime example, as it employs both storage and workspace in an interpenetrating system.

The stove-top and oven are process areas. So is the sink. The dish drainer is a buffer, but one where the process of drying takes place. Kitchen tables and counters may be treated as both workspace and buffer, at the discretion of the user. (And at the initiative of guests. The sink, too, is often used as a buffer, but with very discouraging results.)

The living room, in many houses, is a collection of comfortable body buffers which may also be process areas (e.g. for conversation), a table which tends to hold magazines and drinks (storage and buffering) or be used for other purposes (e.g. a work surface for kit-building). Plus whatever vista pleases the residents, often with decorations symbolic of lifestyle.

The bathroom has process areas for washing and lounging, shallow shelves (cunningly hidden behind a mirror) for parallel access to body equipment and supplies, and, of course, an output buffer for organic material.

The bed is a process area for various bodily processes, though the access structure of what you can reach once you're in it will vary considerably.

Houses ought to have loading docks. Somehow, in the architectural image of the American home, there is still the idea of a little self-sufficient cottage. Yet, in fact, much of our waking lives is given over to the getting, loading, and unloading of objects. This, and not the much less frequent welcoming of guests, should be the focus of the major portal.

(I once lived in a house whose entrance to the basement was in the rear, and whose parking space was three-quarters of the way around, or 150 steps. Pine trees obstructed the short way. A ten-carton loading sequence, then, involved 3000 steps, or about a mile of superfluous trudgery.)

A large part of comedy has dealt with access structure, much of it around the home: Fibber McGee's closet, folding-bed jokes, inter-blocking and cross-prevention, the perils of trying to do several things at once.

Buffering—putting things aside or in piles—is a great problem in human life. Packing and unpacking, handling papers, handling anything. What shall we do with this thing? Put it on the back porch until—

Buffering has various causes, legitimate and less so.

The simplest buffering is when the things we buffer have well-defined destinations, but we don't want to make a separate trip around the house for each. (As when we unpack.) So we toss things in a pile and then walk the pile around. Here buffering is to save time.

Sometimes we buffer because the objects require treatment: bills to be sorted (paid if you're lucky), letters to be answered. And there may not be time to do so. Time-buffering again.

Unfortunately, this is also related to procrastination, which we, uh... won't get to right now.

Sometimes we don't know what we will (or can) do with something. A part may be lacking, or a plan, or the money, or the will. So we buffer out of uncertainty.

Sometimes we buffer for an overview. Getting ready for a trip, for example. "Do we have it all?" can be better answered if we've put everything for the trip in the foyer. Or, "Do we have enough?" when a picnic is imminent and nobody remembers what food is going, so we get it all together where we can see it.

And to sort is to buffer. Every destination pile when you sort is a buffer. (Often you can't have as many piles as the categories you mean to sort into, because you can't toss things far or accurately enough, or the room is too small. So you have to sort into piles which in turn must themselves be sorted. Levels of buffering.)

Project-involved people, who bring work home or have hobbies, need far more buffer space than people who don't. Indeed, projects tend to fill more and more space, making the home areas less and less satisfactory vistawise.

Of special interest is use of the bed as a buffer area. Project-involved people sometimes use the bed as the buffer space of last resort, in effect ransoming it to force completion or clearing of important work in progress—packing tomorrow's briefcase, say, or going through the mail. This represents a commitment that can only be shirked by pushing all the stuff onto the floor, or ignominiously sleeping in it.

There is a fundamental rift between shipshape people and buffering people. The shipshape principle, one approach to access structure, is that everything should be put back in a canonical storage space, in canonical condition, when it is not at zero-order (or first-order) access. This makes a lot of sense on boats, and some people use it as an organizing principle in their lives. But shipshapers are totally incompatible with project people, or any other bufferers.

We've spoken here as if you knew where everything is. How likely is that?

If you have enough shelf (drawer) space, you can store by category. That's great: just like a supermarket, aisle six for soup and canned meats. But the trouble is that you don't usually know at the outset how much there is going to be in each category, so you may end up with a lot of empty space and a lot of crowded space, and have to rearrange just to balance space.

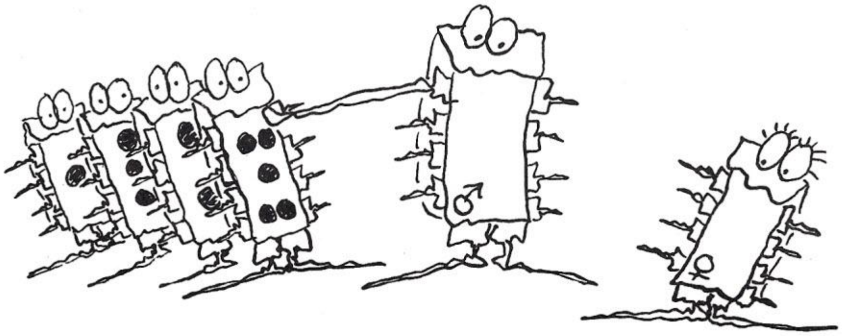
Another problem is that categories get a little sluffy. Consider the supermarket. You have probably never seen frozen worm patties or canned armadillo meat, but you probably know where they'd be if they were there. On the other hand, where do they keep *matches?* Stores do not seem to agree.

This is the problem of *conceptual clarity* of access structure. If you are going to find what you want, access structure has to have conceptual clarity.

It's great to have an eidetic memory, in which case you can just put anything anywhere. But if you don't remember perfectly where you put things, and especially if you are really absent-minded, there has got to be an *idea* of the right place to put each thing. And this idea has to be one you can remember, so you'll know where to look when you want things again.

Combine this with buffering and piling. Piles and buffer

EVE 'N' PARITY



"Watch this, Eve.... Domino Theory."

Cartoon by Korkie

spaces develop a conceptual unity; often you create a pile that has a certain meaning you can't put into words. But you know where it is and what's sticking out of it reminds you of *what* it is. (The visitor to my house is enjoined *never to combine two piles.* Each pile has a conceptual unity, but two randomly combined piles no longer have any meaning at all.)

Moving a pile, or "making it neat," thus has deleterious side effects of the worst kind. As does "straightening up," in the aunt-Harriet sense of rendering all straight lines orthogonal in consonance with the house's principal coordinate system. This destroys information about what is in each pile, which had been so easily seen from what had been sticking out diagonally.

(It is said that Hefner's round-bed editorial chamber is cleaned, first, by *mapping* the position of each object or pile of papers, then vacuuming and making the bed, then returning everything to where it was. Is this what Hefner really needs Polaroids for?)

"Straightening up" has a much more horrifying sense: *putting things in unknown places just so they don't show.* This is a capital crime in my abode.

And this is what leads to true mess. True mess is not how things look, it's a loused-up condition of access structure—exactly the opposite of what many neatness bugs think. Mess is a condition of information loss with respect to access structure.

When you lose information about where things are, it is nearly the same as losing the objects themselves. This leads to an important distinction. *Clutter* is merely what looks unappealing, and is of no concern in this analysis; mess is the presence of objects or collections *whose access structure is unknown.*

(This is another thing wrong with deep shelves and large closets: you don't see what's behind. Information *about* access is lost.)

Now we come to the Great Question.

Wouldn't it be nice to have everything always accessible? Zero-order access of more than one thing is impossible unless you're ambidextrous, or a one-man band. Even first-order access is usually impossible for more than ten things. So the question is rarely what is "accessible," but rather, what things are at what order of access.

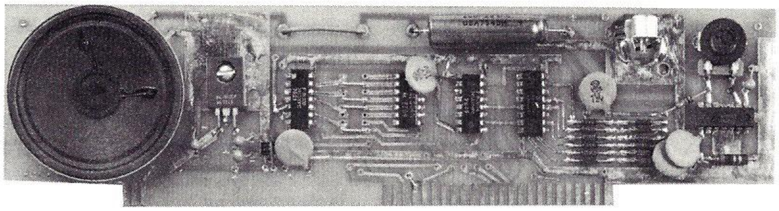
If you had an immense array of shelving—say, a factory building two hundred feet square, with row on row of shelves, grafted onto your present house or apartment—it would make storage quite easy, wouldn't it. But look at the cost. Access costs and costs and costs.

Now, many people are trying to legislate us into keeping less. Architects who offer us cubes to live in, designers (like the authors of *Nomadic Furniture*), artists who have endorsed the throwaway principal (like Warhol and Les Levine). They tell us to keep nothing. So did Thoreau.

Well, that's their bag. I speak for access, for insights to help you keep things conveniently and get at what you keep so you can use it. It's not that we keep too much. It's that we haven't decent access equipment. Meaning, among other things, attics and hallways with good shallow shelves. ▼

NEWTECH
Model 6

MUSIC BOARD



PRODUCES MELODIES, RHYTHMS, SOUND EFFECTS, MORSE CODE, TOUCH-TONE SYNTHESIS, AND MORE!

- S-100 bus compatible
- Jumper selectable address decoding
- 6-bit latching digital-to-analog converter
- Glass epoxy printed circuit board with plated-through holes and gold-plated fingers
- Audio amplifier
- Speaker

$59.95 ASSEMBLED AND TESTED AVAILABLE THROUGH YOUR LOCAL COMPUTER STORE

- Volume control
- RCA phono jack for connection to external audio system
- Complete users manual with BASIC program for writing musical scores and 8080 Assembly Language routine to play them
- 60-day parts and labor warranty

NEWTECH COMPUTER SYSTEMS, INC.
131 JORALEMON STREET * BROOKLYN, NEW YORK 11201 * (212) 625-6220

The Human Factor

IT CAN'T BE DONE

by
Andrew
Singer

In my office there is a table with a red Touch-Tone telephone on it. This telephone is a bit of whimsy. I live on Cape Cod. It is not the ideal spot for a computer consultant, and I don't clean up on the local trade. The red phone is my "hotline" to the rest of the world. At the end of the month, my telephone bill looks like a directory of New England. My regrettable lack of brevity in conversation contributes daily to Ma Bell's solvency. Some days, the phone never rings. Other days would give the casual observer the impression that I am operating a highly lucrative betting parlor.

It was on one of those busy days, several months ago, that a colleague of mine called. The late afternoon sun glared in my eyes as I listened to his exasperated outpourings. Over the last few years, he and I have been pursuing the holy grail of human engineering for interactive computer systems. One of our efforts has been the design of a "considerate" program that would help a person to edit typed material using a computer. Although I had a hand in the design of this creature, I scrupulously avoided having anything to do with the actual programming of it. Other tasks demanded much of my attention, and past experience has taught me that it is unwise to be only partially involved in a programming project.

Occasionally, I heard tidbits about the editor project from my colleague, and these, without exception, made me uneasy. There was nothing definite that I could put my finger on, but anyone who has managed programmers learns to read between the lines and to fear the worst.

Once under way, most programming projects depend heavily on the efforts of one or another key programmer. The very worst that can befall a programming project is for one of these crucial people to disappear. Oddly enough, such disappearances are rather commonplace, and usually reflect the inability of the person involved to achieve some stated objective in the work.

"Oh, that's trivial," says the confident programmer, "I can whip that off in a few days."

But programming problems have a way of exploding, usually after considerable effort has already been exerted. What begins as a few day's work may take months—or worse, the programmer may not be able to arrive at a workable program at all. In these straits, many programmers can think of only one solution: hide.

This was precisely what had happened to the editor project. The lead programmer made some early decisions about the way the program was to operate. He defended these decisions staunchly after he made them. As the program neared completion, he discovered, to his horror, that they were absolutely wrong. It was too late to change the decisions, and the program clearly would not perform adequately. So he dropped out of sight. And returned a few weeks later, much chagrined, with the report that the nearly completed editor program used more memory space than was practical and would run far too slowly.

It was unhappy news. Especially since the project was close to its completion deadline, and this particular programmer was due to leave in a few days. But life must go on, and so a new, "better" programmer was selected to take over the project. Any new programmer is a "better" programmer to someone managing a project that has just disastered, if only because the new person has had nothing to do with the disaster. But such changes in personnel do not necessarily improve matters.

Together, the new programmer and my colleague spent a few days looking over the nearly completed but unsuccessful version of the editor. Owing to the volatility of programmers and their vanishing qualities, little surveys of the wreckage like this one are a common ritual. And just as we say prayers for the departed, there is a litany that goes with the demise of a program. All programmers know it by heart and recite it fondly. It goes:

> This program is no good.
> It's so bad that it's not worth fixing.
> It would be better, faster, easier, cheaper, to rewrite it from scratch.
> Besides, it should have been written in XYZ, in the first place.

(XYZ is the new programmer's favorite computer language, and is always much more suited to the project than whatever language preceded it.)

My colleague is an outstanding researcher, but not a hardened programming manager, and when he heard the litany, he realized that it was time to call in reinforcements. So my red phone rang, and I got to listen to twenty minutes of woeful monologue ending with, "Besides, he thinks it should have been programmed in PASCAL in the first place." I should point out that my colleague has often been on the receiving end of my own sad stories, so I was sympathetic to his plight.

Perhaps, I suggested, it might help if I had a chat with the new programmer to see what difficulties he foresaw in fixing the existing program. This proposal was greeted with enthusiasm, and in a few moments the phone at the other end changed hands. I began my questioning cautiously. At first, it seemed that everything possible was wrong with the program. Indeed, it looked as if the thing was just a collection of loose ends barely tied together and hardly deserving of the title "program" at all. As my questions became more pointed however, it was apparent that things were not so bad and, actually, aside from a few minor flaws and a number of program bugs, there was only one big problem. But, as the new programmer was quick to point out, that big problem permeated the entire program, and there was obviously no easy way of fixing it. At this point there was a pause in the conversation.

Finally, the programmer said, "What do you think we should do?"

I thought for a moment, and then replied, "I think you should fix the existing program."

"But that's impossible. It can't be done."

Instantly, I had a sense of *deja vu.* Those last words were familiar. Very familiar. I was looking down a corridor in time, facing myself, uttering the same words.

It was the first real programming project I had ever tackled. The computing center I worked for had a nifty assembler called SAVE. An assembler is a computer program that enables one to write programs in the computing machine's own language using alphabetic symbols and abbreviations instead of the numeric codes that the machine actually understands. The assembler translates or "assembles" the symbolic program into the coded form that the machine can use. SAVE had been developed locally by a gang at the computing center, and they had blessed it with some unique features.

One of these was called a "relocating, self-loading, magnetic tape format." Dissected, this lovely string of computer jargon amounts to the following: once assembled, the coded version of the program would be saved by copying it onto magnetic tape. To run the program in the computer, it was necessary to copy the program into the computer's memory. The process of copying a program stored on magnetic tape or somewhere else into a computer's memory for the purpose of running the program is called "loading." Loading is accomplished by a special computer program called a "loader." After they have been assembled, programs usually must be loaded into a fixed spot in a computer's memory. This can be a nuisance, especially when a number of programs are to share memory. A "relocating" loader permits programs to be "relocated," that is, loaded anywhere in memory.

When SAVE wrote an assembled program onto magnetic tape, it embedded the program into another program, a special relocating loader. This loader was so tricky that it could relocate and load itself as well as the program it was supposed to load. That was why programs written on tape by SAVE were said to be in "relocating, self-loading, magnetic tape format."

This clever scheme for automatically loading and shuffling programs around memory had one major drawback. Once a program had been translated into machine language and written out onto tape in this self-loading format, it was virtually impossible to change it. Of course, the original symbolic version could be changed, and the program could be assembled again. But for a large program, reassembly could take an hour or more. Everyone agreed that it would be very desirable to be able to make minor changes directly to the coded version if only that tricky self-loading format didn't get in the way.

To loftier minds around the computing center, the solution was obvious. Write a program that would copy the old tape, make the desired changes in it, and write out a revised tape that would be just as relocating and self-loading as the old one. These wise heads pointed out that such a recopying process could be accomplished in a matter of minutes, even for long programs. Clearly, this would do the trick.

The part of SAVE which actually put programs into self-loading format was called the "Loading Routine Generator." The Loading Routine Generator was infamous around the computing center for being one of the cleverest and trickiest bits of programming ever seen. By various ingenious devices, it had been reduced to five hundred computer instructions. This compressing of programs into the smallest possible size was necessary in the early days of computing because memory was expensive and scarce. Unfortunately, the compression process added a remarkable overlay of obscurity to the program's inherent complexity. As it stood, the Loading Routine Generator could defy most methods of cryptoanalysis. It was, in computer jargon, "a nasty piece of code."

Clearly, the Loading Routine Generator was going to figure in any attempt to diddle the tapes it generated. Consequently, no one wanted the job of writing the diddler program. As low man on the staff, I was volunteered for the job. In my ignorance, I was blissfully unaware of the suffering that lay ahead.

The author of the Generator was a friend of mine who shall remain nameless. With his help, I got a printout of the program and settled down to study it. He offered to answer any questions I might have, which seemed generous at the time. Later, when I discovered that I could not even frame an intelligent question for him, I was less impressed. For two weeks I studied, or rather tried to study, the Loading Routine Generator. Actually, I couldn't make head or tail out of it.

Finally, the boom fell. My supervisor on the project invited me into his office. The conversation was brief.

He began with, "How are you doing with the Loading Routine Generator?"

"I can't understand it at all," I said honestly.

"Well, you'll have to use it to do the job."

The two weeks of frustration had been building up in me, and I was beginning to think that the whole thing was an immense practical joke that the entire computing center staff was playing on me.

Confident that this was the case, I said, "That's impossible; it can't be done."

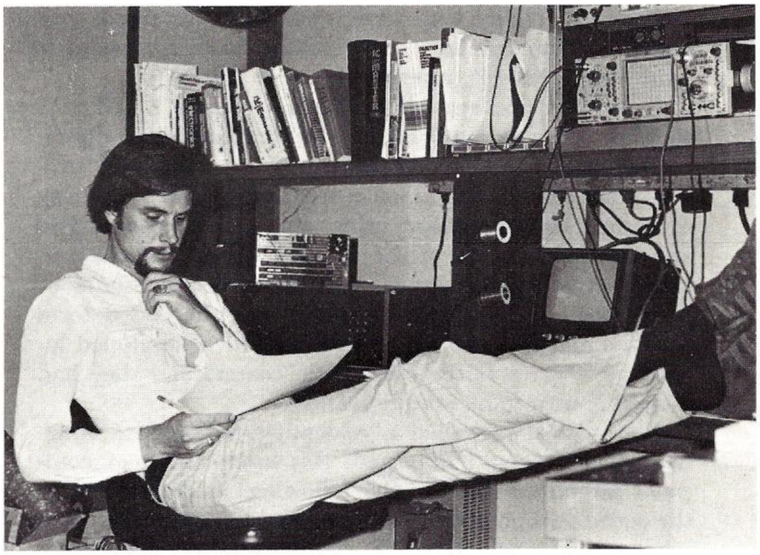
At these words, an odd expression crossed his face and he said, "Nothing is impossible. If you can't do it, we'll find someone who can. Don't come back until you've cracked it."

Of course, he turned out to be right. I simply hadn't been trying hard enough. I don't remember too much about the six weeks that followed our meeting, but they were dismal. Eventually though, the Loading Routine Generator yielded its secrets, and I was able to complete my project.

Since then, I've always been skeptical when a programmer tells me that something can't be done. And that is why my office is painted blue and white, and has a good view of the horizon. I don't ever want to lose sight of that blue-sky-vision that there isn't any task, programming or otherwise, that's impossible, that can't be done.

And the editor project? Well, they decided to take a much harder look at the program, and spent a few weeks examining it in greater detail. They found a lot more problems and programming bugs in it and with clear conscience decided to scrap it and start over again. In PASCAL. Naturally. Personally I remain unconvinced about the project. Even so, nothing really is impossible—even if you have to start over again. ▼

FROM BOMBS TO ROMS



"Why *is* it you do what you do?" I am directing this elementary question to an intense young man who, for the past four years, has chosen to design and build microcomputer systems out of his rustic, self-styled laboratory in the San Mateo hills, overlooking the time-honored countenance of Stanford University. Flashing oscilloscopes blinking hieroglyphics onto a TV screen, pages and pages of plain yet incomprehensible numbers and letters, integrated circuits, and green boards with maze-like configurations covering them—all these objects are his constant companions in his daily struggle, and I wonder why....

Sulfur with the highest purity known to man? Well, it made pretty good gunpowder.

"Because I like what I do," he says simply to me.

Yes, I perceive instantly that he likes it, believes in it so strongly, perhaps with passion, that it is hard to question his motives. I must accept the allure for this man of the all-powerful computer, a machine which seems to mirror man's genius as well as his shortcomings and his frailties.

Jack Glick is a dark haired, quietly serious thirty-five year old scientist and family man, with a wife, two daughters, and more animals than he cares to remember; for a living he runs Modal Systems in Redwood City, California. His interests are as widespread as the California land he loves: an appreciation of all forms of music ("I've actually composed a number of electronic symphonies—well, maybe one symphony and two dirges!"), carpentry ("I'm currently working on a system by which the computer will enable me to saw a straight line...."), gardening ("The computer does the watering, but running the rototiller will take a little longer!"), and studying history ("I see a video-displayed Dymaxion worldmap, dynamically demonstrating the movements of men and nations from the beginnings of civilization ....").

Glick has been a scientist ever since he was a young child, tinkering in his parents' New Jersey basement, concocting such diverse wonders as high altitude rockets, microphotographic equipment, a machine which optically analyzed chemicals, and a sulfur zone purifier, which refined sulfur to the highest purity known to man. Highest purity known to man? What for, I wonder. His answer: "Well, it made pretty good gunpowder!"

Jack was a serious though precocious child, a winner of science awards in high school, then arrested for blowing up a neighbor's gazebo while testing a new explosive. "Thank God I gave up pyrotechnics before the sixties!" He finally achieved a degree in physics after a tumultuous career at a small midwestern college....

"Why are you dropping out of school *again?*" cried his parents.

"I think it was that course in Existentialism, last quarter, or maybe the fact that western man was never meant to learn the Russian language...." While at school, he spent many hours at the local pub, trying to convince his liberal-arts friends that science was not of necessity dull, dry, or incomprehensible, that in fact science should be an all pervasive discipline. "All is one," he used to comment philosophically, "and Physics is One!"

Logically or not, then he moved on to a job in research at Fels Institute in southern Ohio, where he discovered electronics. "Although I had never seen an integrated circuit before, I quickly became proficient at designing mediocre electronic equipment after a year of burning out hundreds of dollars worth of instruments and parts ...."

Two years later he moved westward to Stanford University and a new job at the Medical Center. During his first day on the job there he was introduced

KEY ROUTINES

HEART RATE ANALYSIS ROUTINE
(8080 Machine Code)

```
HR      LXI     H,RDET      Load HL pair with the address of the R wave detection parameter
        IN      EKG         Convert the EKG channel of the A/D converter
        CMP     M           Is the converted value an R-wave?
        RC                  Exit if R-wave not detected
        INX     H           Advance add. pointer to RR interval timer
        MOV     A,M         Read RR interval timer
        CPI     REFRAC      Is refractory period over?
        RC                  Exit if not over
        MVI     M,00        Clear RR interval timer—ACC equals RR interval
        LHLD    RBPNT       Load H,L with add. pointer for RR interval buffer
        MOV     M,A         Save RR interval in buffer
        INX     H           Increment pointer
        SHLD    RBPNT       Save pointer
        MVI     A,RMAX      Get maximum low address for buffer
        CMP     L           Is buffer full?
        RNZ                 Exit if buffer not full
        LXI     H,RBUF1     Initialize RBPNT for next set of intervals
        SHLD    RBPNT
        CALL    BUBLE       Bubble sort RR interval buffer
        LDA     RBMID       Get median RR interval value
        LXI     D,HRDIV     Get dividend
        CALL    DIVIDE      Calculate heart rate from RR interval—result returned in ACC
        LXI     H,HRALRM    Load HL with address of heart rate alarm parameter
        CMP     M           Test for alarm
        JC      HR2         Skip next if no alarm
        OUT     ALARM       Enable bradycardia alarm
HR2     LXI     D,HRDISP    Convert binary heart rate to ASCII and store in heart rate display register
        CALL    BACV        pointed to by registers DE
        RET                 Exit
```

REGISTERS AND CONSTANTS
NEEDED FOR HEART RATE ROUTINE

```
        RDET    00          R-wave detection level (1 byte)
        HRT     00          RR interval timer (1 byte)
        RBPNT   RBUF1       Address pointer for RR interval buffer (2 bytes)
        RBUF1   00          1st address of RR interval buffer (1 byte)
                DS 4        (4 bytes)
        RBMID   00          Middle address of RR interval buffer
                DS 5        5 bytes to last address of RR interval buffer
        HRALRM  00          Heart rate alarm parameter (1 byte)
        HRDISP  DS 3        Register from which heart rate is displayed (3 bytes)
        REFRAC  EQU         Refractory period (1 byte)
        RMAX    EQU         Maximum low address for RR interval buffer (1 byte)
        HRDIV   EQU         Dividend constant for heart rate determination (2 bytes)
```

UTILITY ROUTINES CALLED BY HR

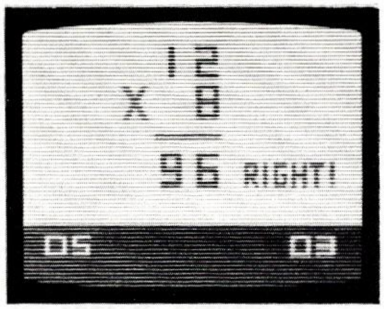
BUBLE	*This routine performs a bubble sort on a list of data, starting at the address in Reg. HL to the low address in ACC.*
DIVIDE	*Performs a double precision divide. Dividend in registers DE, divisor in ACC, and quotient returned in ACC.*
BACV	*Converts binary value in ACC to ASCII values, and loads results starting at the address pointed to by register DE.*
RTC	*A real time clock routine is implied for incrementing RR interval timer.*
DISP	*A video display routine is implied for displaying the heart rate in register HRDISP.*
KB	*A keyboard routine for entering analysis parameters.*

RESPIRATION ANALYSIS ROUTINE
(8080 Machine Code)

```
RESP    LXI     H,RESAD     Load HL with address of respiration detect level
        IN      RESP        Convert respiration channel
        CMP     M
        INX     H           Advance add. pointer to respiration cycle flag
        JNC     RRATE       Jump to determined respiration rate if respiration is detected
        MVI     M,0         Clear respiration cycle flag
        INX     H           Advance add. pointer to alarm parameter
        MOV     A,M         Load ACC with apnea alarm period
        INX     H           Advance address pointer to apnea timer
        CMP     M           Is time since last respiration signal greater than apnea alarm period?
        RC                  Exit if no apnea alarm
        OUT     ALARM       Enable apnea alarm
        RET
RRATE   MOV     A,M         Respiration cycle flag to ACC
        MVI     M,1         Set cycle flag
        INX     H           Advance add. pointer to apnea timer
        INX     H
        MVI     M,0         Clear apnea timer
        ANA     A           Check if last converted value was a respiration cycle
        RNZ                 Exit if same respiration cycle
        INX     H           New respiration cycle
        MOV     A,M         Read respiration period timer
        LHLD    BBPNT       Load HL with add. pointer for respiration interval buffer
        MOV     M,A         Save respiration interval in buffer
        INX     H           Increment pointer
        SHLD    BBPNT       Save pointer
        MVI     A,BMAX      Get maximum low address for buffer
        CMP     L           Is buffer full?
        RNZ                 Exit if buffer not full
        LXI     H,BBUF1     Initialize BBPNT for next set of intervals
        SHLD    BBPNT
        CALL    BUBLE       Bubble sort respiration interval buffer
        LDA     BBMID       Get median respiration interval value
        LXI     D,RRDIV     Get dividend
        CALL    DIVIDE      Calculate respiration rate from respiration interval—result returned in ACC
        LXI     D,RRDISP    Convert binary respiration rate to ASCII and store in respiration rate display
        CALL    BACV        register pointed to by registers DE
        RET                 Exit
```

REGISTERS AND CONSTANTS
USED BY RESP

```
        RESAD   00          Respiration detect level
        CYCFLG  00          Respiration cycle flag = 1, if last value converted was a resp. and it equals 0 if
                            last not a resp.
        AALARM  00          Apnea alarm period
        ATIME   00          Apnea timer
        RTIME   00          Respiration period timer
        BBPNT   BBUF1       Respiration buffer add. pointer
        BBUF1   DS 5        1st add. of respiration period
        BBMID   00          Median respiration period
                DS 5
        BMAX    EQU         Maximum low address for buffer
        RRDIV   EQU         Dividend constant
        RRDISP  EQU         Location from which respiration rate is displayed
```



HOME COMPUTERS

The Products America May Never Know It Needs

by Martin Himmelfarb

Though applications for computers continue to widen on the hobby front, little has been done to push the home computer closer to commercial reality. Technologically, we're ready for the personal computer, but from a marketing standpoint we might just as well still be counting with our fingers and toes. Unless the entire marketing structure of the personal computer industry is changed, drastically and soon, the computer will never become the ubiquitous household servant it was hoped to be.

Part of the problem lies with our captivation by the technology itself and the mystique it breeds. People with their first computer, for example, are disparagingly termed "first time users" by digital savants and made to feel like naive initiates to opium addiction. Computer hobbyists, deservedly or not, have the image of seeming more concerned with the intricacies of *Star Trek* than with the everyday job of getting along in life. And many manufacturers of small computers continue to couch their sales literature with esoteric jargon only a graduate engineer could love.

Meanwhile, the millions of consumers who could benefit from computers in their homes and small businesses remain unreached, except perhaps by the distrust they have developed for that stereotyped elitist, the computer hobbyist.

If this present situation continues, the American consumer may well bend over backwards to make sure that a computer never enters his home.

The American consumer may well bend over backwards to make sure a computer never enters his home.

In a mass marketing context, the estimated twenty-five thousand computers purchased this year for home and small business use are miniscule. Even by 1980, when industry analyst Alan Kaplan predicts annual sales of fifty-five thousand computers, the

Photographs courtesy of Fairchild

numbers will still be too small to entice the large-marketing oriented firms, which think only in terms of millions and which have the power to make the personal computer a widespread reality.

As it stands now, the only hope for the personal computer, from a mass marketing viewpoint, is the programmable television game.

Programmable TV games, already on the market, are the products closest in characteristics to the personal computer. As opposed to the predictions of personal computer sales reaching a paltry fifty-five thousand in 1980, programmable TV game sales are expected to top nine million. When they see numbers like these, big consumer electronics firms take an interest and make every effort to determine the product characteristics that are necessary to appeal to the largest number of potential buyers.

The marketing problems to be solved for programmable TV games closely parallel those for personal computers, so at this point I will diverge into current consumer electronic marketing philosophies. If you are outraged by the crass nature of what follows, remember that it doesn't apply to you personally. Rather, it applies to that vast sea, somewhere out there, of American consumers who collectively spend one-half trillion dollars each year on what economists loosely call "durable" goods.

To appeal to the greatest number of people, programmable games must be affordable, easy to use, capable of accepting a large number and diversity of readily available programs and, in general, they must offer some tangible benefit to the consumer.

As in television, the most obvious benefit is amusement. But as has been proven time and again, a solitary amusement soon becomes boring and the game falls into disuse. For this reason, games manufacturers are thinking along broader lines and foresee the programmable game developing into the wider role of a home entertainment system centered around the TV set and capable of attracting all family members.

Such a home entertainment center would provide games that range from the simple ping-pong type to those like chess that require strategic planning on the part of the player. In addition, these microprocessor-based systems could provide an educational service with programs that present information and then test the user's knowledge with multiple-choice or true-false quizzes. This is similar in many respects to the evolution of the record player, which commercially made its debut as an entertainment medium but soon evolved as an educational tool—for example, to teach languages.

A bit of sexism, the housewife test, is the crucial factor in marketing home-oriented consumer products.

Similar evolution of the programmable TV game all hinges on the availability of cheap, durable programs in large numbers. In turn, the programs all hinge on the availability of cheap, reliable program-loading peripherals and input media. The game manufacturers are still searching for the best ones, and to date have only made tentative short range decisions, which, they admit privately, are subject to change at a moment's notice.

+,−
+,−
x
x,÷
6
FAIRCHILD
VIDEOCART-6

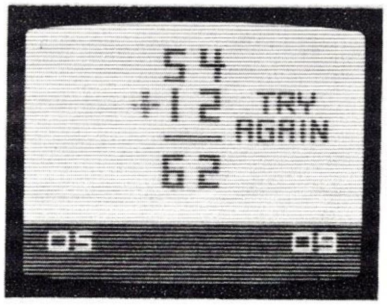
Selecting an input medium for a programmable game requires the consideration of three basic parameters: storage capacity, durability, and cost, all of which bear on market acceptability. (Input data rate is unimportant for the game-and-entertainment market sector.)

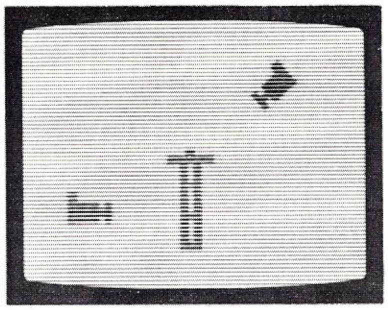
Storage capacity: In their search for the best input medium, manufacturers have devoted a great deal of thought to program length. Opinions are still divided, but thought on the matter seems to have centered on about 1K bytes as the maximum marketable program length. Games above this limit, many manufacturers believe, will simply be too complex to have wide market appeal. At the same time they feel that 1K imposes no great restriction on the diversity and number of programs available.

Programs stored on calculator data strips, for example, in wide use by Texas Instruments and Hewlett-Packard, are generally less than two hundred bytes long. On the other hand, at the extreme end of the game spectrum, an unoptimized BASIC language program for *Star Trek* requires 8K bytes, and a streamlined version can easily fit into 4K bytes. But *Star Trek* is a relatively complex game, perhaps too difficult to play to pass the mass marketing "housewife" test.

This bit of sexism, the housewife test, is thought to be a crucial factor in the successful marketing of a home-oriented consumer electronic product. While fathers will buy complex toys for their kids, the pundits say that mothers will not. The feeling of the mass marketers is that any device with more than four or five controls will make a housewife uncomfortable and thus lower sales. In the TV games market, the major purchasing segment is said to be housewives buying products on behalf of their husbands and children. So if they are to pave the way for personal computers, programmable TV games had better be easy to use. Closely parallel to this requirement is the need for durability.

Durability: A game that easily breaks will not be replaced. It will be exchanged for another product, returned for a refund, or simply thrown





in a closet to gather dust. Because it has no practical value, and because alternative forms of amusement are readily at hand, there is little pressure for a family in having it repaired. So if programmable games are to become pervasive, they must be durable and reliable.

The same goes for the program-loading devices and storage media. They must be able to withstand rough handling, dirt, humidity, and heat, as well as stray magnetic fields from other nearby electrical equipment. In short, they must survive storage in a Death Valley attic, an Alaskan cellar, and, worst of all, handling by a three-year-old child.

Besides withstanding physical abuse, loading devices and media must be resistant to accidental data errors. A scratch on an audio recording is nothing more than a petty annoyance, but a single bit error in a program can render a game inoperative.

Cost: Absolutely no doubt exists among manufacturers that the retail price of a program must be very low compared to the price of the game machine itself. This is analogous to the hi-fi business where relatively low record prices—$3.98 to $7.98—lead to eventual sales of perhaps one hundred LPs for each record player.

Computers must survive storage in a Death Valley attic, an Alaskan cellar, and, worst of all, handling by a three-year-old child.

Similarly, in the games business, many programs must be made available cheaply if player interest is to be sustained, and if the TV home entertainment center is to evolve into the personal computer. Taking this analogy one step further, with game machine prices at about the same level as those of record players, big manufacturers would like to see programs at about the same price as LPs. Only in this way, they feel, will player interest be sustained and will non-game programs penetrate the market.

Beyond the problems of data storage capacity, ease of use, durability, and cost are the intangible aspects of market acceptance. Products directed toward mass markets must be promotable both to retail outlets selling them and to consumers buying them.

One important key to promotability, and thus sales, is the consumer's preconception of the product. Consumers may, for example, have a negative preconception of the hobby computer because of its apparent complexity, cost, and mystique. On the other hand, they seem to have a positive preconception of the TV game, as evidenced by ever increasing sales. So, to penetrate mass markets, computer manufacturers must capitalize on and extend the concept of the programmable TV game. Program media manufacturers must do the same.

FAIRCHILD
VIDEOCART-4

An example of this strategy is the way Fairchild packaged the read-only memories (ROMs) that store programs for its video games. The company cleverly designed the package to look like a tape cartridge, a product consumers are comfortable and familiar with, and it was displayed in stores in a familiar way. However, the price of the ROM cartridge is high, and a Fairchild official admits that because of

RAINBOW COMPUTING, INC.

Supplier of

WAVE MATE
THE DIGITAL GROUP
DEC PDP
Computer products

Peripherals and Supplies from

PERSCI
CENTRONIX
DIABLO
MAXELL
COMPUTER DEVICES
LEAR-SIEGLER
MULTI-TECH
TEXAS INSTRUMENTS

Specialists in Design, Implementation
and Support of
Custom Hardware/Software Systems for
Business, Educational, and Personal Use

Experts in most major computer
software including
CDC, IBM, PDP
BASIC, COBOL, FORTRAN, PL1
LISP, SIMULA, SNOBOL, SPSS, BMD's
COMPASS, MACRO,
6800, & Z80 assembly languages

10723 White Oak Ave., Granada Hills,
Ca. 91344
(213) 360-2171

Reconditioned Teletypes
Guaranteed 90-days on return basis
ASCR 33 $950 KSR 33 $595

Free delivery within 25 mile
radius of New York City.
Shipping extra beyond radius.

IMSAI edge connector and guides:
ten for $39, $4.50 each.

COMPUTER MART OF
NEW YORK
118 MADISON AVE
NEW YORK, NEW YORK 10016
(212) 686-7923

THE COMPUTER STORE

Featuring

MITS Altair
Data General Micro Nova

Also your headquarters for

Proto Typing Equipment
Wire Wrapping
Magnetic Supplies

"Where personal computing begins."

THE COMPUTER STORE
63 SOUTH MAIN ST.
WINDSOR LOCKS,
CONN. 06096
(203) 627-0188

BACK ISSUES

Complete your collection NOW!
Back issues $3.00 postpaid from
ROM Publications Corp.
Route 97
Hampton, CT 06247

ROM

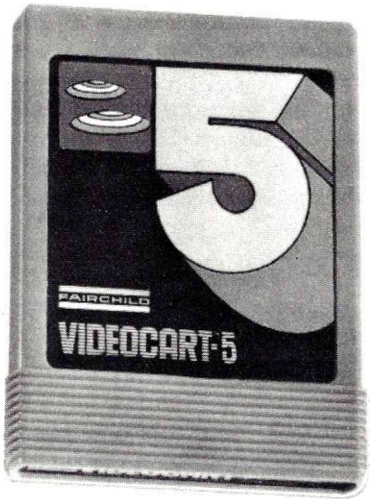
Solitary amusements soon become boring and games fall into disuse.

the price he expects that game buyers will each only buy four or five program cartridges a year.

There are alternatives to the ROM cartridge, each with its own set of marketing assets and liabilities. Traditional computer input peripherals, because of the original intent of their designs, fare the worst. The floppy disk drive holds more data than required and costs too much; the cassette tape is closer to pricing requirements, but cannot withstand rough handling. (On the plus side for the cassette tape is that it has already been mass marketed and has reached a point of acceptance as an audio medium.) The calculator data strip is partially accepted. But, while inexpensive, it stores too little data and, like the tape cassette, is susceptible to physical abuse.

Until recently overlooked as a potential game and personal computer input medium, the magnetically striped credit card is durable, cheap, and widely accepted by American consumers. Tens of millions of these cards are in use, but in their present form do not store enough data. However, one company, Vertel, Inc., has found a way to squeeze 1K bytes onto a single card that has four magnetic stripes, two on each side.

"Durable enough to scrape ice from frozen windshields, inexpensive enough to be produced for pennies, the magnetically striped card is certainly capable of emerging into new applications as a data entry and program loading medium," says Vertel president Gus Toda.



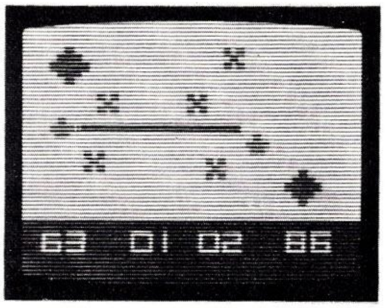
Because the card can be "made for pennies and sold for dollars," Toda feels that it will at once give consumers a wide choice of inexpensive programs and allow manufacturers the price mark-up they need to stay prosperous.

Other digital media may also find applications in programmable games and in personal computers. But mass marketers prefer to deal with proven products and are unlikely to risk an entire product line on an untried device.

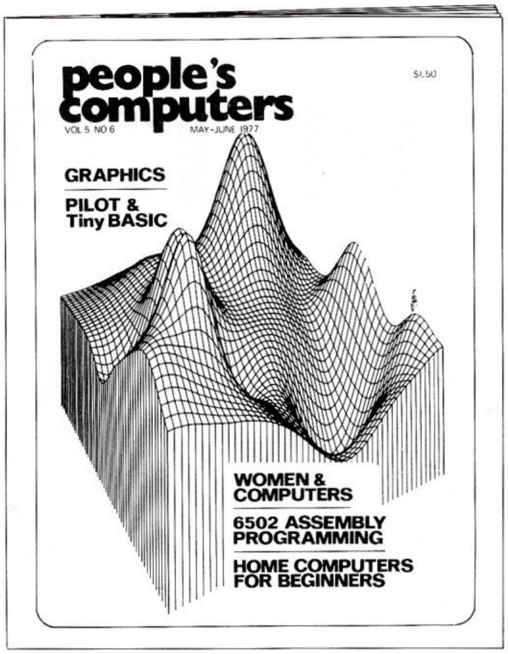
Money spent on programmable TV games, of course, is really being spent on computers. They both contain central processors, memory, input peripherals, and displays. The major difference lies in emphasis: games are application oriented, computers are product oriented. The other difference is in programs. Users can write their own programs with computers, but are limited to supplied programs for TV games.

Computer manufacturers should learn from this that if they want to penetrate mass markets, they will have to deemphasize the niceties of product-oriented technology and start concentrating on applications. And like the CB radio makers who learned the hard way not to put CB jargon in their ads, computer makers will have to start speaking English.

The actual differences between programmable TV games and home computers may be only semantic, but billions of dollars are at stake—not to mention the answer to the question: Will home computers ever stop being closet toys and become the useful adjunct to twentieth-century life that is their true mission? ▼



WE'RE DIFFERENT!



people's computers

$1.50

VOL 5 NO 6

MAY-JUNE 1977

GRAPHICS

PILOT & Tiny BASIC

WOMEN & COMPUTERS

6502 ASSEMBLY PROGRAMMING

HOME COMPUTERS FOR BEGINNERS

No Ads

No Frills

All Content

People's Computers is for anybody and everybody who wants to learn more about computers, how they work and how to use them. We aren't padded with color advertisements, we're cover to cover with articles, listings, reviews, games, letters and interviews.

Read the whys and wherefors of different computer languages. Discover what a 'chip' is, how computer networks operate and what new products are available. Learn to program from our series on easy-to-use computer languages. Find out how computers are used in education, in robotics, and for communication by the handicapped. Each issue also contains programs with extensive comments, that you can use on your computer at home, at school or at work.

people's computers

Please start my one year subscription (6 issues) to *People's Computers* and bill me for $8.

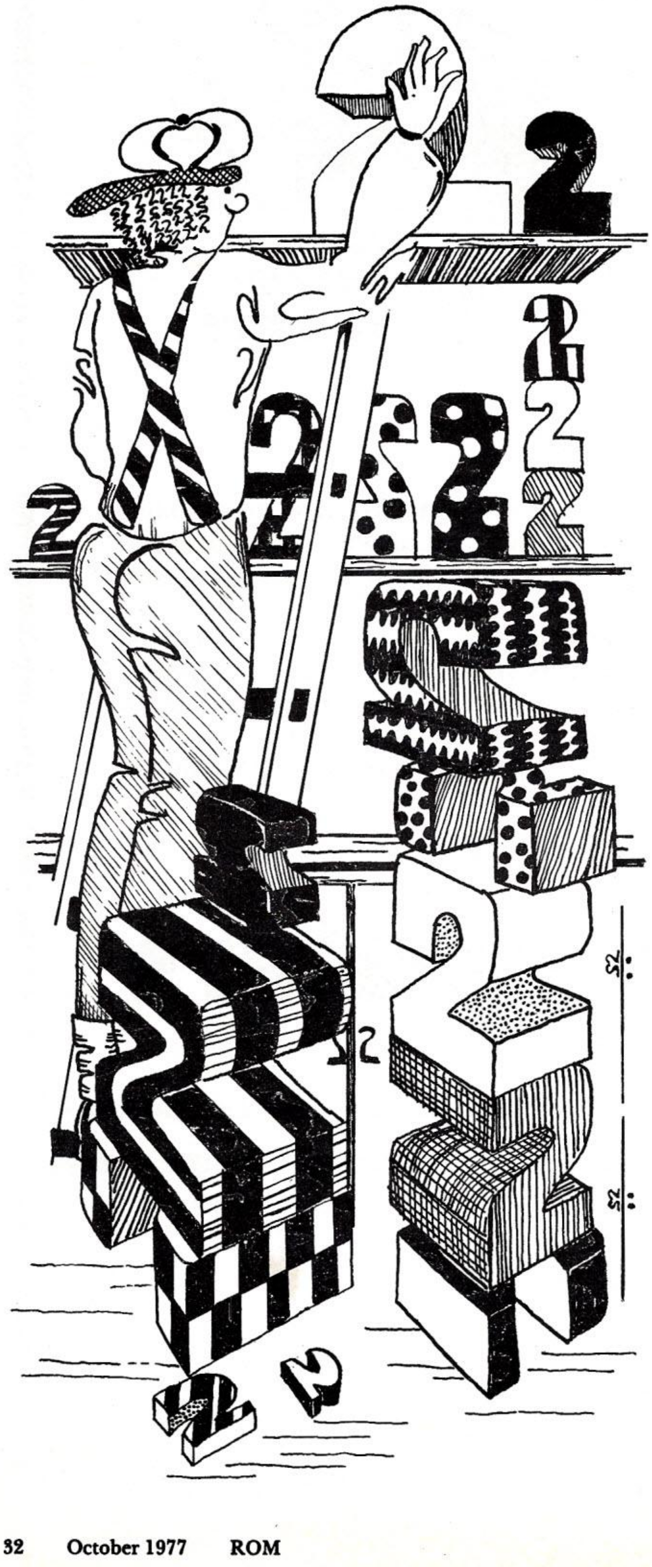
NAME ____________________

ADDRESS ____________________

CITY ____________________ STATE ____________ ZIP ____________

Mail to: *People's Computers*, Dept 52, 1263 El Camino Real, Menlo Park, California 94025

PUTTING TWO &



Binary

by Tom Pittman

How does a computer do arithmetic? The answer depends on how curious you really are about the subject.

On the first level are those who need to know about computer arithmetic so that they can use it to solve problems with their computers. Maybe you do not like mathematics at all, but it seems necessary to know something about the way computers do it so that you can properly command the machine. If you are in this group, read to the end of this page (and maybe halfway into the next), then skip to the next article. I'm not kidding. You do not need to know anything at all about binary arithmetic to use it in your computer. As long as you stick to BASIC (which is better than machine language any day), there is absolutely no necessary reason for you to understand how it does its internal arithmetic. In fact, some BASIC interpreters do not use binary at all in their arithmetic. So, in getting to know your computer, spend your first efforts on understanding the logical operations that a program needs to go through. As we saw in the August *ROM* ("Software—The Genie In the Bottle"), the computer is not so much a numerical calculating machine as it is a logic processor. Come back at your leisure and read the rest of this article.

The second level of readers are those who would like to know how binary arithmetic works, but do not necessarily feel the urge to do it themselves. I am fascinated by watching machinery work. I am all thumbs when it comes to building something mechanical that works, but I want to understand the principles involved anyway. If this is your attitude towards computer arithmetic, read on. Don't

Illustrated by Linda Smythe

TWO TOGETHER

Arithmetic Explained

get bogged down in the numerical examples sprinkled through these pages, but keep an eye out for the general principles. Skim over the details.

The third category of reader is the one who hopes eventually to write his own binary arithmetic routines. I will not give specific machine language instructions in this article, but I hope you will gain enough basic understanding of the way things work so that you can write your own machine language routines. And please forgive the elementary stuff at the beginning; not all of our readers are as sophisticated as you are.

I hope by now that most of my readers are in the second level. The first level, take your cue and skip now. And while I intend to tell you exactly how to do it, I hope there are not many of you who really are shooting for the third level. This is because if everybody spends his effort writing arithmetic routines, who will do the interesting things like robots, music, art, games, voice, and so on? Let's not everybody make (invent) wheels.

The primary function of a number—at least the positive numbers—is to indicate how many things you are talking about; this is true whether the things are apples, as in "ten apples," oranges, cows, dollars, inches, degrees Fahrenheit (you didn't know you could think of a degree of temperature as a "thing," did you?) or batting averages (in this case, the things are hypothetical balls hit in a presumed thousand tries). So with a one-digit number, you can count up to nine things (9), or you can say there are none (0). If we put two of these one-digit numbers together it becomes a two-digit number. The right hand digit means the same as a one-digit number, but the left hand digit represents how many tens of things. In other words, the things that the left hand digit counts are really groups of ten things. The number 57 refers to five groups of ten things and seven individual things. The grouping is of course imaginary. If we put a third digit in front of this number, it counts groups of hundreds of things. We call this positional notation, and the digits are said to be "weighted." The right hand digit has a weight of one, the second digit (moving to the left) has a weight of ten, and so on. You can think of it as, "each object counted by the second digit weighs ten times as much as each object counted by the first (i.e. rightmost) digit"—if we are talking about 57 pickles, each of the five groups of ten pickles weighs about ten times as much as each of the seven individual pickles (assuming, of course, that they are about the same size).

I am going to concentrate in this article on binary numbers. Some computers operate with binary-coded-decimal numbers, so I will say a little about that next month. "Binary," "decimal," "octal," and "hexadecimal" are terms which refer to number systems. Or more specifically, to the radix of a number system. The radix of a number system has to do with the number of different values a single digit can take. In decimal, the radix is ten, corresponding to the ten digit values (0,1,2,3,4,5,6,7,8,9) which may occupy a single digit position. In binary the radix is two, and a single digit position may take only two values (0, 1). In the mathematical systems that we all know and love, it happens that the arithmetic procedures are the same regardless of the radix being applied. In other words, the same steps you go through to add two decimal numbers, you also go through to add two binary numbers. We will exploit that fact.

In positional notation, such as we have inherited from the Arabs for our decimal numbers, the position of each digit is weighted by a power of ten: one, ten, one hundred, one thousand, etc. This is because ten is the radix. In binary numbers each digit is also weighted by successive powers of the radix, in this case two: one, two, four, eight, etc. Note the correspondence, shown in Table 1:

Table 1

Power	Tens	Twos
0	1	1
1	10	2
2	100	4
3	1000	8
4	10000	16
5	100000	32
6	1000000	64

Notice, however, that I cheated. All of those numbers are shown in decimal. Look at the same table slightly modified (Table 1a), in binary:

Table 1a

Power	Tens	Twos
0	1	1
1	1010	10
10	1100100	100
11	1111101000	1000
100	10011100010000	10000
101	...	100000
110	...	1000000

Two things you should notice in Table 1a. First, the "Twos" column in binary looks exactly like the "Tens" column in decimal. Each entry is a one with some number of zeros (possibly none) to the right of it; the actual number of zeros is exactly the number in the Power column. We *read* the Tens column as "one, ten, one-hundred, ..." and the Twos column as "one, two, four, ...," but in both cases we *write* (in the respective number system) "1, 10, 100, ..." And in both cases a zero means *no objects,* and a one means *one object* of the appropriate weight. If it seems I am belaboring this point, it is because if we grasp the similarities and differences, we will intuitively know what we can borrow and what we need to change.

The second thing I wanted you to notice is the actual numbers in the binary version of the Power column. As you can see from the decimal version, these are counting from zero to six. Let us examine this counting a little bit more closely:

0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
...	...
19	10011
20	10100
...	...
99	1100011
100	1100100
101	1100101
102	1100110
...	...

In both cases we started at zero, then counted to one. In both cases, when the rightmost digit reaches the highest possible value, the next count takes it back to zero and counts the next digit to its left; if that digit is also at its highest value, it also goes to zero and we count up in the third position; and so on. The only difference is that in binary the highest value a digit can take is one, so we have to bump the next digit much more often than in decimal which can go all the way to nine before the next digit is affected.

When the computer is going to do addition, we ask it to "memorize" a binary addition table, an example of which is shown in Table 2.

Table 2

	0	1
0	0	1
1	1	10

Actually, the hardware has this table built in, so we do not have the problem of coaxing the computer to learn it. To add two one-digit numbers, you look up one of them along the top and the other along the left margin. Then, drawing a straight line along the respective column and row, where they meet is the sum. To add two numbers of more than one digit each you simply add the rightmost digits, then add the next pair of digits (to the left), and so on. If at any time you get a two-digit sum, you call the left-hand "1" of the sum a "carry," and add it to the next sum. Some examples:

$$\begin{array}{rrl} & (1\,1) & \text{(carries)} \\ 57 & 57 & \\ \underline{+21} & \underline{+69} & \\ 78 & (16) & (7+9, \text{ carry } 1) \\ & (11) & (5+6, \text{ carry } 1) \\ & (12) & (11+1 \text{ from previous carry}) \\ & 126 & \text{(correct answer)} \end{array}$$

An easy binary example:

$$\begin{array}{r} 1000 \\ \underline{+\ \ 101} \\ 1101 \end{array}$$

Another, with carries:

(1 1)	(carries)
110	
+ 111	
1	(1 + 0)
(10)	(1 + 1, with carry)
(10)	(1 + 1, with carry)
(11)	(10 "two" + carry from previous digit)
1101	(correct sum)

Now, most of the computers built today do not add one digit at a time. Instead they have special logic wired into the hardware which adds eight or sixteen digits (or "bits," ostensibly a contraction for "BInary digITS") in one fell swoop. They do this with what is called a "parallel adder" and a "lookahead carry generator" which I must confess, even I do not completely understand—I just use it. Nonetheless, the sum is exactly the same as if it were done one bit at a time.

If you take the largest number of any particular number of digits (say three: 999) and add it to the largest number of the same size, you will never get more than 1 for a carry out of the sum: 999 + 999 = 1998. You might get no carry out (as in 111 + 222 = 0333). But adding two numbers will never have a carry of more than one. Try a few numbers to convince yourself.

The computers you will be working with will probably have a "word length" of eight bits. That means that the hardware adder is able to add an eight-digit (binary!) number to another eight-digit number, to get an eight-digit sum. But there is a problem. An eight bit number can count any number. But if we add 11111111 + 11111111 we get 111111110, which is a nine-bit number. The ninth bit (the leftmost one) is called the "carry out" of the sum, and may be used for a variety of purposes, as we shall see. For the convenience of the hardware, we define every sum of n bits + n bits to have an n + 1 bit sum; the carry out may be either 1 or 0.

One of the functions of the carry out of a sum is to signal overflow, or a result too large for the system to handle. Obviously nine bits is too large for an eight bit word to handle, so if the carry out is one, we may consider the addition to have overflowed. This is exactly equivalent to getting an answer too large for your pocket calculator. After doing the sum, the program could test for this condition by examining the carry out bit (there is a special machine language instruction to do this).

For the moment let us ignore that carry out. Look at the following additions:

$$\begin{array}{r} 01010111 \\ +\,01101011 \\ \hline (0)\,11000010 \end{array}$$

$$\begin{array}{r} 11000010 \\ +\,10010101 \\ \hline (1)\,01010111 \end{array}$$

$$\begin{array}{r} 10010101 \\ +\,01101011 \\ \hline (1)\,00000000 \end{array}$$

In the first example we added 87 + 107 = 194. The second example adds 194 + 149 = 87, plus a carry out. Actually the sum is 343, but we are ignoring that carry. The third example adds 149 + 107 = 256, or if you ignore the carry out, zero. What we have discovered is a way to subtract.

If you wanted to subtract 6 from 9 in grammar school, you looked it up in your memorized addition table: you examine each number in the 6 row until you find 9, then look up to see what number was at the top of the column; it is 3, so therefore 9 − 6 = 3. Effectively you are asking, "What number do I add to 6 to get 9?" To subtract 9 from 6, you found that 6 does not occur in the 9 row, but 16 does. You call this a "borrow," and it works like a carry.

If we want to subtract 107 from 194, we ask, "What number, when added to 107, gives 194?" In high school algebra you learn that A − B is equivalent to A + (−B), so we might also ask, "What number, when added to 194, gives 87?" The answer to this last question should be the number "−107," and the third example shows, as we would expect, that the number we think might be equal −107, when added to +107, gives zero. So there you have it. −107 = +149. Some quick algebraic manipulation and you have proved that 0 = 256. As long as we are willing to agree that 0 and 256 are the same number (and they are, for eight bits), then:

$$-107 = 0 - 107 = 256 - 107 = 149$$

The rule is, the difference between a number and some power of two greater than the word size is the negative of that number.

Now that we know what negative numbers are, we can talk about subtraction in the machine. It is nothing more nor less than adding the negative. Most computers have a machine language instruction which automatically computes the negative of a number, then adds it. It is called a subtract instruction. But how does it find that negative?

As we have seen, the negative of a number x is that value which when added to x gives a sum of zero. Let's look at the process for, say, 01010110:

$$\begin{array}{r} 01010110 \\ +\,10101010 \\ \hline 100000000 \end{array}$$

It is not clear how I got the number 10101010 (I sprang it on you without explanation). Consider instead a slightly different sum:

$$\begin{array}{r} 01010110 \\ +\,10101001 \\ \hline 011111111 \end{array}$$

This one has the curious quality that there are no carries out of any of the digit positions. In fact, we have selected a number which has the characteristic that the sum of each bit column is exactly 1, with no carry. Looking at our binary addition table (Table 2), we see that this will be true if we choose a 1 where the original number has a 0 and a 0 where the original number has a 1. The hardware to do this is very simple. But alas, the sum is not the zero we asked for. Fortunately, it is not far away:

$$\begin{array}{r}11111111\\+00000001\\\hline 100000000\end{array}$$

So the number that the hardware has given us is only one less than the negative we really want. This is indeed how the hardware does the subtraction: it "complements" the number to be subtracted (i.e. substitutes 1 for 0, 0 for 1), then adds this complement plus 00000001 to the number it is subtracting from. Let us look at a few more examples:

$$\begin{array}{r@{\qquad}r}01101001 & 105\\-01010110 & -\ 86\\\hline\end{array}$$

is the same as:

$$\begin{array}{r@{\qquad}l}01101001 & 105\\+10101010 & +170\\\hline (1)00010011 & 275=19+256=19+0=19\end{array}$$

$$\begin{array}{r@{\qquad}r}01010110 & 86\\-01101001 & -105\\\hline\end{array}$$

is

$$\begin{array}{r@{\qquad}l}01010110 & 86\\+10010111 & +151\\\hline (0)11101101 & 237\ ???\end{array}$$

But what do we do with the 237? Clearly 86−105 = −19, not 237. But notice also that 19+237 = 256 = 0. In other words, 237 and −19 are the same number (to our binary machine). How can we tell? Well, there is that carry out we have been ignoring. If the carry out of a subtract is 1, the result is positive; if it is 0, the result is negative. Caution: many computers play games with the carry when they do a subtraction, so you need to be careful. We will go over this problem a little later.

One of the problems we have not looked at yet is the question of when a number is negative. How do you tell the difference between 86−105 = −19 and 86+151 = 237? If you remembered, you can say "I added in one case and subtracted in the other." Better, we would like to say that whenever we see 237 it always means −19. You recall that the eight bit number can take 256 values, from zero (00000000) to 255 (11111111). Suppose we select a different set of 256 values, say from zero to +127 (which is 128 different values), and all their negatives. To minimize confusion, we let the positive numbers be the same as the binary numbers we have been using. In other words, 00000000 is zero, 00000001 is one, 00001010 is ten, 00110010 is fifty, and so on to 01111111 which is 127, the largest positive number. For negative numbers we run each of the positive numbers through the hardware subtract instruction, subtracting it from zero, and look at the result. Negative zero is zero, which is as it should be. Negative one is 11111111, because 255+1 = 256 = 0. Negative ten is 11110110, and so on. Negative 127 is 10000001 (check: 10000001+01111111 = (1)00000000). We have used up all the numbers between 00000000 and 11111111 but one: 10000000. What is it?

If you look at all the positive numbers and compare them to all the negative numbers, you will notice a distinctive feature: the leftmost bit of all the positive numbers is 0 and the leftmost bit of all the negative numbers is 1. We sometimes call this the "sign bit" since it tells us the sign of the number. Many computers have special machine language instructions which look at this bit to decide if a number is positive or negative. Now for that orphan number, 10000000. Because the leftmost bit is 1 we can easily assume that it is negative. Since it is one less than 10000001 (which is −127), we call it −128. But what about +128? Obviously +128 = 0−(−128), but if we tell the computer to subtract 10000000 from 00000000, it comes back with 10000000 (try it!). −128 then is an anomaly. There is no +128. We call this method of writing negative numbers "Two's Complement Notation." I suppose it derives from the fact that when you complement a number, you subtract each bit from 1. This is called "One's Complement," and the Two's Complement negative is one greater than the One's Complement.

Earlier I said that the carry out of a sum could be used to tell if the sum overflowed. Now I am going to tell you that it does not work, at least not with signed numbers. Consider some examples:

$$\begin{array}{r@{\qquad}l}01010111 & 87\\+01101001 & +105\\\hline (0)11000000 & +192=-64\ \text{(overflow)}\end{array}$$

$$\begin{array}{r@{\qquad}l}01010111 & 87\\+00011110 & +\ 30\\\hline (0)01110101 & +117\qquad\text{(no overflow)}\end{array}$$

$$\begin{array}{r@{\qquad}l}01010111 & 87\\+11100010 & +(-30)\\\hline (1)00111001 & +57\qquad\text{(no overflow)}\end{array}$$

$$\begin{array}{r@{\qquad}l}10101001 & (-87)\\+11000000 & +(-64)\\\hline (1)01101001 & -151=256-151=+105\\ & \text{(overflow)}\end{array}$$

Clearly, the carry out of the addition does not tell us whether the sum overflowed. We know that the sum of the two positive numbers should be positive, and the sum of the two negative numbers should be negative; if it is not, that is overflow. We can deduce that the sum of a positive number and a negative number will produce a result which is between them, so it cannot overflow. So if we add two numbers of the same sign and the result is not also the same sign, we have overflow. If the numbers do not have the same sign, or if the result does, then no overflow has occurred. When subtracting, since we "change the sign and add," we look at the signs after changing the sign (i.e. finding the negative), but before adding. Some computer hardware will do this for you and remember whether the result is in overflow. Usually however, there is a simpler way for the hardware to determine overflow: during the addition you compare the carry out of the seventh bit (second from the left) to the carry out of the most significant bit; the addition has overflowed if and only if the two carries do not match. I won't go into the mathematical proof that these two methods are equivalent, but you can convince yourself by trying a few numbers.

Suppose we want to deal with numbers greater than 127. The procedures are the same as when in decimal we want to deal with numbers greater than 9: you add one column, then transfer the carry to the next column addition. Only here "one column" means a whole eight bits, and the carry out of the first eight bits is added into the next eight bit partial sum, and so on as necessary. You ignore anything that resembles sign and overflow indications except for the most significant column (or byte). Example:

```
(0)        (0)        (1)              (carries)
   00110100 01010110 10000111
 + 10011010 00010010 11001011
(0)11001110 01101001 01010010
```

In this example we added the rightmost two bytes which generated a carry out of 1. Then we added the middle pair of bytes with the carry in of 1, producing a new carry out of 0. Finally, the most significant pair of bytes are added with a carry in of 0 and a carry out of 0. We look at the sign on the leftmost byte and conclude that we have added a negative number to a positive number to get a negative result, with no overflow. When you go to do this sum on a computer, you find that there is a version of the ADD instruction which also adds in the previous carry.

To subtract several bytes of numbers (we call it "multiple precision" when a number is larger than one ordinary computer number, in this case eight bits) works almost exactly the same as multiple precision addition. First you subtract the rightmost pair of bytes. This gives the rightmost byte of the result. It may require a borrow from the next byte pair if you are subtracting a larger number from a smaller number. A borrow is exactly the same shape and size as a carry, but it is a different color. When the hardware adds the negative of the subtrahend (the number being subtracted) to the minuend (the number being subtracted from), the carry out of the sum is the complement of the borrow into the subtraction. In other words, if the carry is 1 the borrow is 0; if the carry is 0 the borrow is (—)1. Actually this is only a way of speaking about it, because when you get around to subtracting the second pair of bytes, the machine complements the subtrahend, then adds it plus the carry to the minuend, and the answer comes out right. Alternatively, you can think of it as, "You add the negative of the subtrahend, minus the borrow, to the minuend." Let's do it both ways:

```
   00010010 00110100
  -01010110 01111000
```

Doing the right half,

```
   00110100
 + 10001000
(0)10111100   carry=0, borrow=1
```

Now the left half, using the carry:

```
        (0)            (carry)
   00010010
 + 10101001            (complement, not negative!)
(0)10111011 10111100   negative result
```

Now the borrow version:

```
       (-1)            (borrow)
   00010010
 + 10101010            (negative in this case)
(0)10111100
 + 11111111            (add -1 of borrow)
   10111011 10111100   same result
```

All of the computers that have an "add with carry" also have a "subtract with borrow" instruction, which does this automatically. What you have to be careful about is that some of them use the carry flag in the computer to mean borrow when they are doing subtraction, so they can do it the second way; others use the first method, and the carry flag is always a true carry out. If the borrow method is built into the hardware, then you cannot program the computer to simulate subtraction by adding negative numbers, because the borrow flag will be backwards.

Another thing I should mention here in passing relates to the way the carry (or borrow) is added into the sum. While we have been talking exclusively of adding two numbers at a time only, the hardware is actually capable of adding three—as long as the third number is limited to either 0 or 1. So when we use the "add with carry" instruction, the computer adds both numbers and the previous carry in one operation. Similarly the "subtract with borrow" instruction adds the minuend, the complement of the subtrahend, and the previous carry (or the complement of the previous carry if the borrow method is used) in one operation. Instead of actually computing the negative of a number for subtracting, it adds the complement of it, with a forced carry in of 1 (which is the same as no borrow). You will not notice this until you try to follow the logic for subtracting zero from some number:

```
   01010111
  -00000000
```

If it took the negative first and added, you would see:

```
   01010111
 + 00000000
(0)01010111    (no carry out)
```

But actually it does it this way:

```
         +1    forced carry for subtract
   01010111
 + 11111111    complement of zero
(1)01010111    (carry out)
```

The answer is the same; only the carry out is affected. This is necessary so that if you are subtracting zero in the least significant part of a multiple precision problem, the carry into the next bytes will be correct. Of course if the borrow method of subtraction is used, the carry out of the sum is complemented before being stored in the carry flag of the computer.

How about multiplication and division? Multiplying a number by ten consists simply in moving the digits to the left one position, then inserting a zero at the end. Dividing by ten consists of moving the digits one position to the right. In this case you can call the rightmost digit (which is deleted) a "remainder," or you can put a decimal point on it and call it a fractional part. Let's ignore fractions for the time being. 235 divided by ten is 23 with a remainder of 5. 235 multiplied by ten is 2350. Notice that multiplying by ten adds one digit, and that dividing by ten removes one digit. The easiest thing to do is to multiply or divide by two. This is simply moving the binary digits left (or right) one position and inserting a 0 bit. You can also multiply by two simply by adding a number to itself:

$$X \times 2 = X + X$$

Multiplying or dividing by two is not very interesting of itself, but we can use the operation (called "shifting") to do real multiplication and division. Let us begin with the multiplication table:

Table 3. Multiplication Table

	0	1
0	0	0
1	0	1

Not very exciting, is it? But that is the key to using it on the computer.

Take a sample multiplication, three times four. In binary that is:

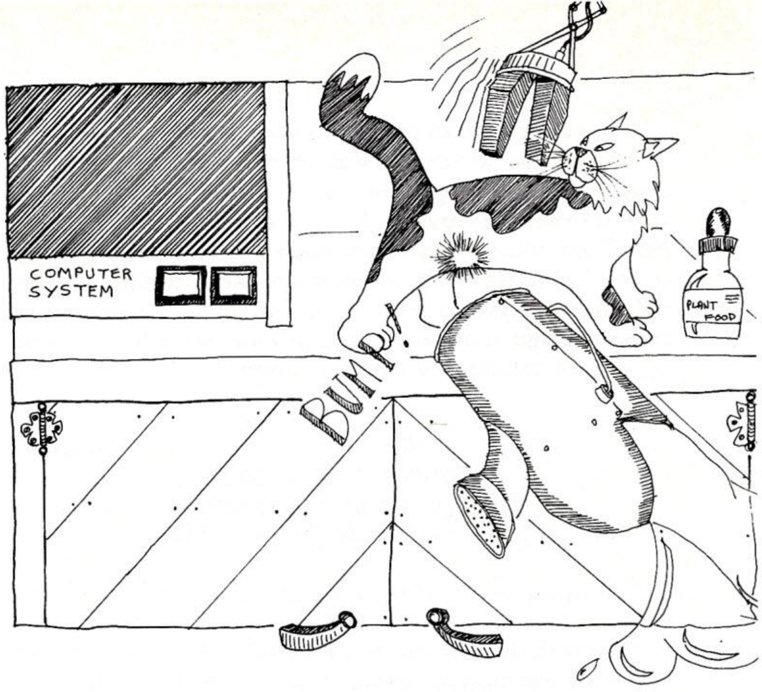
$$11 \times 100 = 11 + 11 + 11 + 11 = 100 + 100 + 100 = 1100$$

But a better way is to take the long multiplication approach:

```
   100
  ×11
 (000)
 +100
+1000
  1100
```

That was a little trivial. Let's try a more complicated one:

```
        01010111       87
      × 01101001     ×105
      (00000000)
      + 01010111
     + 000000000        (87×0×2)
    + 0000000000        (87×0×4)
   + 01010111000        (87×1×8)
  + 000000000000
 + 0101011100000
+ 01010111000000
+ 000000000000000
0010001110101111       =9135
```



As you look at this multiplication, you will see that each partial product (the lines that are added) is either entirely zero, or exactly one times the multiplicand, shifted over by some increasing number of places. It is zero if the bit in the multiplier we are working on is 0 and it is the multiplicand if the bit is 1 (because one times any number is that number). This is easy to mechanize (i.e. it is easy to tell a machine how to do it). It is just a shift and add, repeated eight times, except that you only add if the bit in the multiplier is 1. Actually, rather than trying to add ever-widening numbers, we leave the multiplicand where it is and do the shift after adding each line. This is a little hard to show on paper, but I'll try. Start with three numbers—the multiplier (105), the multiplicand (87) and the product (initially zero):

Multiplier	01101001
Multiplicand	01010111
Product	00000000 00000000

Notice that we have allowed sixteen bits for the product, even though in the previous computation it only used fourteen bits. Now do the following three steps eight times:

1. Look at the rightmost bit of the multiplier. If it is 1, then add the multiplicand to the left hand eight bits of the product. Don't lose the carry out!
2. Shift the product right (i.e. divide it by two), filling in with the carry out of the sum in step 1, or with 0 if you did not add.
3. Shift the multiplier right, and throw away the remainder (the bit you shifted off the right end).

When you have done this eight times, the product will be in the proper place, and the multiplier will be gone (used up). Let's watch it happen:

Step 1. First time through:

Multiplier	0110100(1)
so add Multiplicand	01010111
Product	+ 00000000 00000000
New Product	(0) 01010111 00000000

Cartoon by A. D. Libin

Step 2.	
New Partial Product	00101011 10000000
Step 3.	
New Multiplier	00110100 (1)

Step 1. Second time through:	
Multiplier	0011010(0)
Multiplicand	01010111
Partial Product	00101011 10000000
No addition	
Step 2.	
Partial Product	00010101 11000000
Step 3.	
New Multiplier	00011010 (0)

Step 1. Third time through:	
No addition	
Step 2.	
Partial Product	00001010 11100000
Step 3.	
New Multiplier	00001101

Step 1. Fourth time:	
Add Multiplicand	01010111
Partial Product	+00001010 11100000
New Product	(0) 01100001 11100000
Step 2.	
Partial Product	00110000 11110000
Step 3.	
New Multiplier	00000110

. . . . and so on. You Level three readers can finish this out if you like.

Notice that by shifting the multiplier right each time around, I can always look at the same bit position in the number. This is much easier than requiring the computer to look at a different bit position each time. Notice also that nothing is done with the zeros in the right half of the initial product, except to shift them off the end. Usually the multiplier is put into this space so that it can be shifted at the same time as the product (in other words, we combine steps 2 and 3). This also saves space in the computer. If you look carefully, you will notice that the interesting part of the multiplier is the right-hand end, and the left-hand end is ignored; the interesting part of the product is the left-hand end, and the right-hand end is ignored. Also there are exactly as many interesting bits in the multiplier as there are ignored bits in the product, and vice-versa.

Some people program the multiplication to add to the right-hand end of the product and shift left. This works equally well, but it is a little more complicated because the carry out of the addition may extend into the second of the product. Also, you look at the leftmost bit of the multiplier to decide if to add, and you have to shift before you add instead of after. Other than that, it is substantially the same procedure.

Some computers have hardware to do the multiplying operation; they usually use exactly this method, or some variation on it, which is why the multiply instruction usually takes several times longer than an add instruction. Most of the microcomputers that hobbyists have access to do not have a hardware multiply, so we have to write a program something like I described here to do it. The program of course goes much slower than hardware multiply would.

Division we handle in the same way as multiplication. The simple-minded approach (suitable for small numbers only) starts subtracting the divisor from the dividend until the dividend is less. To divide (binary) 1100 by 100,

$$\begin{aligned} 1100-100&=1000 && (1)\\ 1000-100&=100 && (10)\\ 100-100&=0 && (11) \end{aligned}$$

The number of times you successfully subtract is the quotient.

It is easier, however, if you do it one digit at a time, as in decimal long division:

$$\begin{array}{r} 11\\ 100\overline{)1100}\\ \underline{-100}\\ 100\\ \underline{-100}\\ 0 \end{array}$$

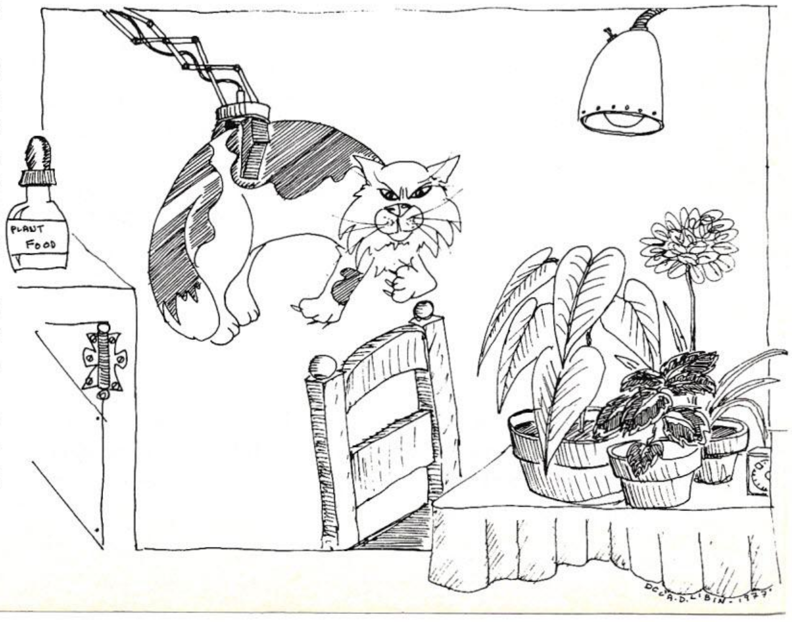
Notice here that you find the first group of bits that is larger than the divisor. Since in binary you can have at most a quotient digit of 1, you put that down, multiply it times the divisor (i.e. copy the divisor under the first group of bits), and subtract it off. Bring down another digit and repeat.

Let us mechanize this a little more formally. Start with the same kinds of numbers you used for the multiplication. Only this time the dividend is the double-length number, and the quotient is initially zero:

Quotient	00000000	
Dividend	00100011 11110100	(9204)
Divisor	01010111	(87)

Repeat the following three steps eight times:

1. Shift the quotient left, multiplying it by two.
2. Shift the dividend left, multiplying it by two.
3. If the divisor is not greater than the left half of the dividend, subtract it from the dividend and add one to the quotient.



Let us follow this through for a few repeats:

```
Step 1. First time through:
  New Quotient        00000000
Step 2.
  New dividend        01000111 11101000
Step 3.
  01010111 is greater than 01000111; do nothing.

Step 1. Second time:
  New quotient        00000000
Step 2.
  New dividend        10001111 11010000
Step 3.
  Dividend            10001111 11010000
  Divisor            -01010111
                     ------------------
  New dividend        00111000 11010000
  New quotient        00000001

Step 1. Third time:
  New quotient        00000010
Step 2.
  New dividend        01110001 10100000
Step 3.
  Subtract divisor    01010111
  New dividend        00011010 10100000
  New quotient        00000011

Step 1. Fourth time:
  New quotient        00000110
Step 2.
  New dividend        00110101 01000000
Step 3.
  Nothing happens.
```

...and so on. Again I leave it to my more ambitious readers to finish out the loop. The last time we will see:

```
Step 1. Eighth time through:
  New quotient        01101000
Step 2.
  New dividend        10011100 00000000
Step 3.
  Subtract           -01010111
                     ---------
  Remainder           01000101 (00000000) (69)
  Final quotient      01101001
```

So the remainder is what is in the left half of the former dividend. There are a few problems to watch out for: If, when you shift the dividend left, you get an overflow (the bit shifted off the left end is 1), remember that fact and force the subtract in step 3. If the left half of the dividend is not less than the divisor before you start the process, you are in trouble and the answer will be wrong. This is called a "divide fault." This is because the quotient is likely to have as many digits as the difference between the number of digits in the dividend and the divisor (the same fact as that the product has as many digits as the sum of the number of digits in the two factors). If the divisor is too small, there will be more digits in the quotient than you can fit into an eight-bit number. A special case of this problem occurs when the divisor is zero. If the divisor is not zero, you are guaranteed a valid result if the left half of the dividend is zero. Unlike the multiply, the divide only works with left shifts.

When hardware designers build a divide instruction into their computers they usually take one or two shortcuts. Since a compare to see if the divisor is larger usually consists of a subtract and a look at the sign of the result, they do the subtraction as if the divisor were not larger. Then if the dividend goes negative, they add it back on; if it does not they add one to the quotient. This is called "trial subtraction." If you write a program to do division in a computer that does not have a hardware divide instruction you would probably do it the same way. Another technique which makes the division go a little faster is called "nonrestoring" because if the dividend goes negative when you subtract, you do not add the divisor back on (restore it); instead you add the divisor the next time around instead of subtracting. You continue to add each time the dividend is negative, and whenever it goes positive you go back to subtracting. The quotient gets ones whenever the add or subtract leaves a positive dividend, and zeros whenever the dividend is negative. This is a little tricky to program, but if you need raw speed, it is faster.

So far we have talked about multiplying and dividing positive numbers only. More often than not, when signed numbers are to be multiplied or divided in computers (whether hardware or software), the signs are compared, and both numbers are set to positive; the computation is performed, and then if the previous two signs did not match, the result is set to negative. This is unfortunately an awkward way to handle signs, and clever people are continually finding ways to get around it.

One of these techniques (for multiplication) relates to the fact that a product normally has twice as many digits as the factors, and that often we do not expect actual products to exceed the number size we are working with. If we are dealing with eight bit numbers (-128 to $+127$) then the product of two of them should, by rights, be sixteen bits. However, perhaps we have no use for sixteen bit products, and we happen to know that they will not occur (not necessarily a reasonable assumption unless you have carefully screened the values being processed).

Assume that n and m are positive numbers, and we want to multiply n by $(-m)$. If you will pardon the algebra,

$$\begin{aligned} n \times (-m) &= n \times (256-m) \\ &= 256n - nm \end{aligned}$$

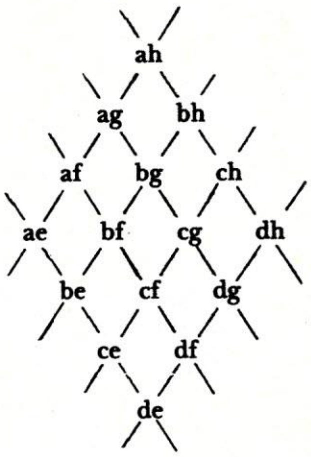
Since $(-nm)$ is the answer we want, we are wrong by $256n$. But anything multiplied by 256 is effectively shifted left eight places, so if we look only at the low eight bits, $256n = 0$. Thus, using two's complement numbers, if we are interested only in the product being the same size as the factors, we can ignore the sign in the multiplication and the sign of the product will be correct.

More often, we really do want a true double-length product, so a technique known as Booth's Algorithm is used. It requires that we look at the least significant bit of the multiplier and also its previous value (before the last shift; initially assumed to be zero). If both bits are the same (either 00 or 11), then you just shift the numbers and advance to the next cycle. If the previous bit was 0 and the present bit is 1, then you subtract the multiplicand from the partial product. If the previous bit was 1 but the present bit is 0, you add instead. When you shift the partial product right, if the sign bit is 1 (negative), you shift in a 1 to keep it negative. This is called "sign extension" or "arithmetic right shift," because the sign remains the same.

There are two ways to do multiple precision multiplication and division. If you have to write a program to do the multiplication or division because your computer lacks the hardware, then it is probably easiest simply to use the same technique, spread out to the appropriate number of bits. That is, you use multiple precision adds and shifts, and you increase the number of times through the loop (the number of repeats) to reflect the number of bits in the multiplier or quotient. So if you want four-byte, thirty-two-bit numbers, you will go through the loop thirty-two times instead of eight; you will carry four-byte sums instead of one-byte sums, and the product or dividend will be eight bytes in all which need shifting. You of course realize that the computer has instructions like the "add with carry" which "shift with carry," so that the bit shifted off the end of the previous byte is shifted onto the front of the next byte.

The other approach is most useful if the computer has hardware multiply and divide instructions. Given that you have a multiply that gives you a two-byte product of two one-byte factors, you can get the eight-byte product of two four-byte factors (abcd times efgh) thus: Compute ae, af, ag, ah, be, bf, bg, bh, ce, cf, cg, ch, de, df, dg, and dh. Arrange them (mentally) into a diamond as shown in the diagram opposite.

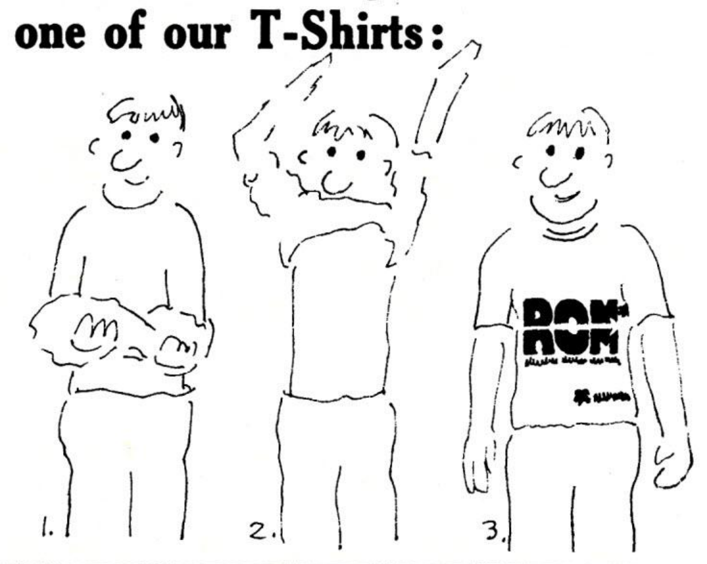
Then, starting at each of the open ends pointing up, follow a zig-zag path down, adding alternately the left and right bytes of the product terms. Start at the right, and the least significant byte of each sum is the next byte of the full product. Thus the first sum consists of the right half of dh only, and it is the least significant byte of the product. The next sum is the right half of ch + the left half of dh + the right half of dg; its right byte is the second byte of the product. The third sum starts with the left half of the second sum (the carries out), and adds the right half of bh + the left half of ch + the right half of cg + the left half of dg. And so on, for eight sums. If this looks hairy, that is because it is hairy. I suppose that when you have more than four product terms to deal with, it becomes easier to treat each product term as a term in long multiplication—in other words, work it like the radix of the number system is 256, not 2. Then you will be shifting bytes over instead of bits, but the procedure will be the same.



For division you must treat the operation as a radix-256 long division: you divide the first two bytes of the dividend by the first byte of the divisor, then multiply the quotient times the whole divisor and subtract from the appropriate number of bytes of the dividend. If it goes negative you reduce the quotient byte and repeat the multiply. The difference from the subtraction is the beginning of a new quotient (you bring down the next byte), and you repeat. I haven't tried to concoct an example to show how it is done, because I don't think you will have any need to do it.

Coming up in *ROM* we will see how microprocessors *seem* to compute in decimal while *actually* doing it in binary. We will also see how to change numbers from binary to decimal and back, both in your head and in the computer's logic unit. And no study of computer arithmetic would be complete without a discussion of fractions and floating point numbers. Or, as we sometimes say, "You ain't seen the half of it yet!" ▼

This is how to get into one of our T-Shirts:



ROM*
COMPUTER APPLICATIONS FOR LIVING
* Read Only Magazine
ROM Publications Corp., Route 97, Hampton, CT 06247

Name
Address
City
State Zip
☐S Qty. ____ ☐Child S Qty. ____
☐M Qty. ____ ☐Child M Qty. ____
☐L Qty. ____
☐XL Qty. ____
☐Check or money order enclosed. $5 ppd.
☐Master Charge ☐BankAmericard 2 for $9 ppd.
Exp. date Card#
Please allow 4 to 6 weeks for delivery.

The WONDERFUL DREAMS of DOCTOR K

by Hesh Wiener

In Sands Point, New York, on suburban Long Island, a stone's throw from the IBM Country Club, stands a complex of modern buildings surrounded by wooded rolling hills. The buildings of the Helen Keller National Center are tan and white, the lawns are green, the sky, jutting past New York's smog bubble, is quite blue. The road that passes in front of the center's wrought iron fence is curved and narrow; its traffic is sparse and moves slowly. Although the area is quite populous, the remnants of what must have been uniformly thick forest are still full of life. If you pause you can hear birds gossiping in the trees, and if your eyes are quick you will see the small animals that live in the shrubbery.

It is a beautiful place to work, and Doctor K works here. His clients, who also know how beautiful this place is, arrive at that conclusion by entirely different means from yours and mine. For the most part, the people served by Frederick M. Kruger, Ph.D., can neither see nor hear.

Up to fifty persons at a time live at the Helen Keller National Center for Deaf-Blind Youths and Adults. They may stay for eight weeks to be evaluated by the hundred-person staff, or they may live at the center for years, developing skills that will permit them to give and take and share. When they first arrive, some of the Center's clients are prisoners in their own skulls. Doctor K, with help from the computer, is trying to set them free.

As the director of research for the Helen Keller Center, Doctor K has devoted a lot of his time to the development of three machines that could enable the deaf-blind to participate more fully in a world that takes sight and hearing for granted. His dream machines are the Telebraille, the Institutional Wrist-com and the Residential Wrist-com.

The Telebraille is a portable terminal by which Braille can be sent over telephone lines, either between two people or between a person and a computer. The Institutional Wrist-com is a two-way communication system that can keep the Helen Keller Center's clients in touch with the staff, with each other, and with the outside world. The Residential Wrist-com is a system for monitoring a home or apartment environment that might otherwise confound or even prove dangerous to a deaf-blind person.

In addition, Doctor K presides over Interchange, the association of people who use Interdata minicomputers. He also participates in professional societies, writes and presents numerous papers, tunes in to ham radio, plays golf, and takes flying lessons. He pilots a pretty hefty computer rig: a sixty-five kilobyte Interdata 70 (much like that at the Helen Keller Center) hung with dual floppies, a Teletype and a couple of CRT terminals, including one homebrew glass Teletype.

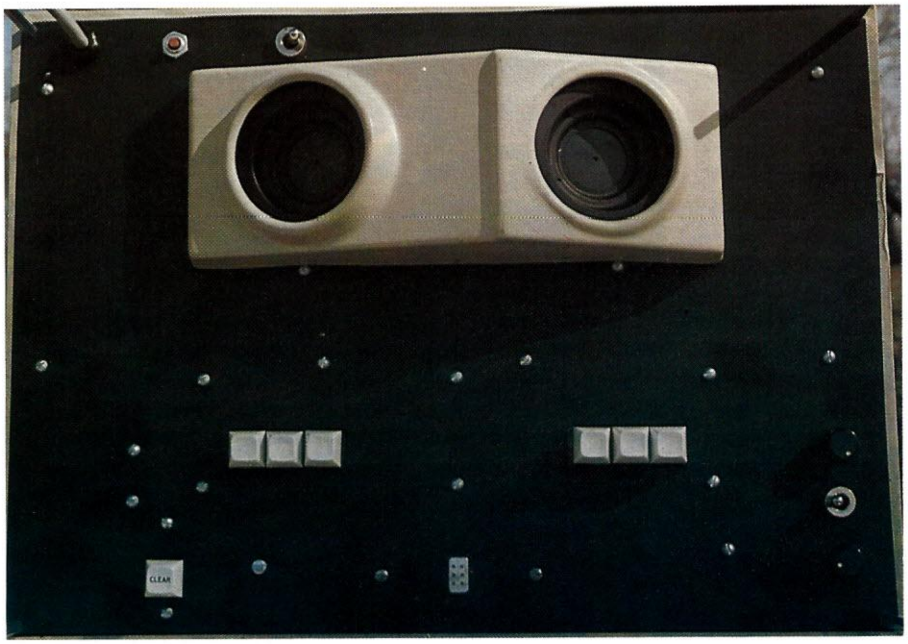
Is Doctor K kidding? I wondered as he told me about himself. How does he do it all?

Here's how:

"Let me show you around this place," said Doctor K, breaking into a fast walk. "We can get more done if we talk and look at the same time. And stop smoking in the lab."

I found out later that the smoke would probably set off an alarm in the computer room, sending thirty pounds

Photographs courtesy of Dr. Frederick M. Kruger



If you or I are feeling a little lonely, we can pick up the phone and dial a friend. Someone both deaf and blind and a little lonely can pick up the phone and dial. . . and then? Where before the presence of interpreters has been required, now the deaf-blind can make a telephone call in complete privacy. The computer-age device making this possible is the Telebraille, developed in the research department of the Helen Keller National Center for Deaf-Blind Youths and Adults. In much the same way as a teletype communicates by telephone with a computer, audio-frequency-shifted digital signals keyed to the Braille alphabet can be sent back and forth. Note the six dots at lower center on the Telebraille, the Braille touch receiver.

of inert gas down on the machinery and any people who happened to be inside. There is some speculation that Doctor K got the fire extinguisher because he dislikes smoking, but everybody who really knows him thinks he wouldn't be so subtle.

When they first arrive, some of the Center's clients are prisoners in their own skulls.

Tagging along with Doctor K as he chugged down the halls, I found our intense conversation was frequently interrupted by the greeting of staffers (who speak English), deaf clients (who speak sign language), and deaf-blind clients (who speak a special touch language).

To say that I was astounded by the scene would grossly understate my reaction. The Helen Keller Center is simply amazing, particularly at a trot with the ebullient Doctor K reeling off explanations, anecdotes, and observations.

"This is the gym. You'll notice that the floors curve up near the walls. That's to keep people who play active sports, like basketball, from running into the walls. Here, try to walk on this narrow beam."

I could.

"Now try it with your eyes closed."

I couldn't, but apparently Doctor K's clients can.

"In here is a complete apartment."

It was immaculate.

"When you can't see, you have to be extra careful about cleanliness. It's also harder to cook."

Doctor K told me that the deaf-blind test water with a wooden spoon. If it's boiling, you can feel the vibrations. The Residential Wrist-com will eventually include a probe, so a deaf-blind person won't have to watch every pot.

"Why shouldn't a deaf-blind person be able to cook complicated meals?"

I don't know, I thought, realizing that I would have thought otherwise fifteen minutes earlier.

"Here's the machine shop."

ROMtutorial ROMtutorial

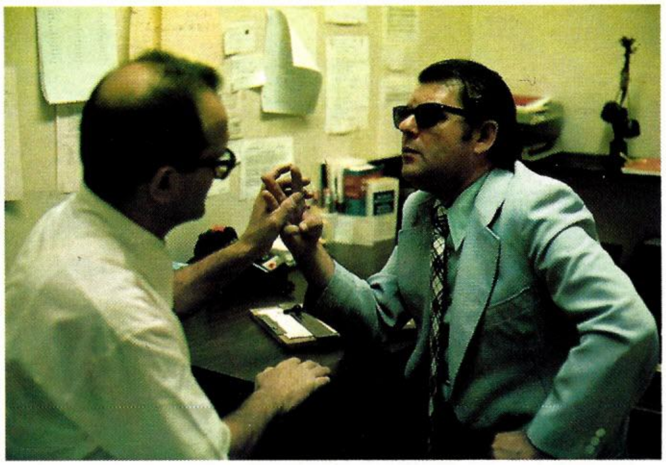
Minicomputer: A table-top size computer, usually somewhat faster than a microcomputer and always more expensive.

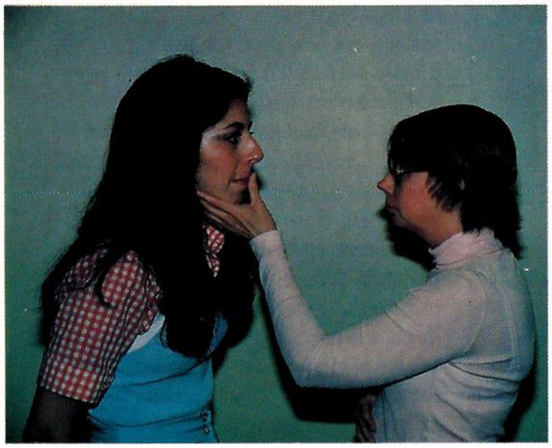
Kilobyte: 1000 (in decimal) or 1024 (in binary) bytes (eight-bit units) of data.

Floppy: A floppy disk is an information storage device which uses a disk that looks like a forty-five rpm disk, but is coated with a magnetic surface like the surface of a recording tape. The disk or flexible "floppy" should not be removed from its dust jacket.

CRT: Cathode ray tube. A television-style display device.

The essential art of finger spelling is shown in this conversation between Mr. Robert Prause, a placement specialist at the National Center, and a deaf-blind client. Will some small significant machine soon be able to aid the sense of touch in face-to-face communication with the deaf-blind?





How do you talk to someone who can neither hear you nor see your lips move in words? Here TADOMA, tactile lip reading, is demonstrated by Ms. Linda Hirsch, audiologist at the Helen Keller National Center, to Ms. Tommie Goins, a deaf-blind client who herself plans to become a professional rehabilitation aide working with the deaf-blind.

I flashed on my surgeon cousin, who can see and hear, is not clumsy, but who caught his hand in a planer not long ago.

The work lying near the machines was damn good. The shop had regulation saws, drill-presses, lathes, and grinders. There was no sign of carnage.

Smart is knowing that this is a world for people.

"Some of our clients are really good at this stuff. They'd help us build things, I think, but there is a lot of red tape in the way. We can't pay them, and we can't not pay them either. Anyway, let's go on to things I understand better."

Trot, trot. That's how Doctor K gets everything done. He runs a lot. And he's not a small guy.

I first met Doctor K a couple of years ago through Phil, a mutual friend, who is maybe twice the good doctor's size, and who also has an aura like a klieg light. They both grew up in New York. Sometimes they hang out with Howie, another son of the City of Menschen, but smaller, even more quick-witted and always in harmony with his friends who love Chinese food.

Phil and Howie told me, in case I didn't figure it out, that Doctor K is smart. To Phil, who is a government muckety-muck and computer whiz, who flies a mean glider, fences, and performs legerdemain, "smart" means more than books. Same goes for Howie, who is a punster, songwriter, and uses his M.D. as a way to fill a chair at an Ivy League med school and as a means to some end in cancer research. To these guys, smart is perceptive, smart is fast, and smart is knowing that this is a world for people.

Doctor K really enjoyed showing off the Helen Keller National Center. The outdoor walkways, he pointed out, have posts along one side. These posts have low-glare lighting that points down from about waist height for the partially sighted.

"If the posts were on both sides, a client could easily become disoriented. By using a cane properly, it is easy to get around this place."

The gym floors curve up so people who play active sports won't run into the walls.

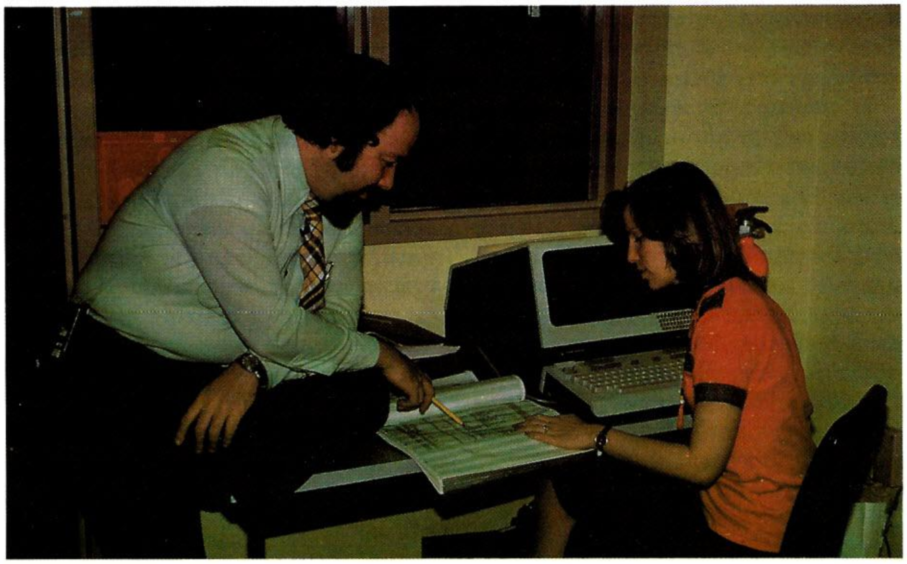
The path that winds around the roller skating rink and jogging track has many different surfaces. The Center's clients have to learn to get around on all kinds of terrain.

Doctor K switched channels as we wrapped up the tour, discussing the elevators that have buttons which push back when you've reached your floor, the problems of adapting computer technology to the jobs at hand, and the other people who work at the center.

He is, he explained, just one more scientist in a field of dedicated people. His predecessors used the best technology of their eras to aid the deaf and blind, and he is doing more or less the same thing. Today's most fruitful technology is electronics, particularly digital electronics. The microcomputer represents not only a terrific way to build new machines for the deaf-blind, it is a way to bring older devices up to date or to get the old and the new to coexist.

Telecommunications systems have also been used by the deaf for some time, and the communications standards that were first established by the pioneers of the field are in use today. Model 28 Teletypes, which communicate via the five-level Baudot code, comprise an extensive network. Newer equipment, which uses the seven-level ASCII code most computers speak, requires circuits that can translate old codes into new ones if they are to be most useful. In addition, there is now a need for more general code-translating machines that will permit electronic discourse among people using ASCII, Baudot, Touchtone, and Morse codes.

"A deaf-blind person can learn to



Doctor K
(Frederick M. Kruger),
here with programmer and
research assistant Ms. Barbara Schor,
in a rare moment—he's not running.

read Morse code very well. You send it to a buzzer that is placed against the skin. Frequencies of 200 to 300 Hertz are the most effective," according to Doctor K.

Morse code may be effective for some deaf-blind persons, but Braille is an even better way to provide information to people who can neither see nor hear. That's why Doctor K has been developing a machine that can send Braille over telephone lines.

His device is the Telebraille, a portable system for sending and receiving information in a form useful to his clients. It's been in the works for a while, and Doctor K thinks he has most of the problems under control now.

It is possible for you to build a Telebraille at home, with the exception of one part: the electromechanical Braille cell. To build that component, you need a machine shop. High schools and colleges have facilities that are more than adequate to do the job.

The electromechanical Braille cell is actually very simple, but it has to work perfectly for the Telebraille to be useful. It consists of a metal plate with six holes in it. The holes are arranged in two columns of three, like the dots on a die's six face. The dimensions of the cell must match that of an embossed Braille character.

In each hole is a pin with a rounded tip. The diameter of each pin is that of a dot on a Braille character. The pins are each connected to solenoids which, when energized appropriately, push up the heads of the pins to form a single Braille symbol. The symbol may be read by a blind or deaf-blind person in the same way an embossed Braille symbol on a piece of paper is read.

The Telebraille, then, is a machine that can receive encoded data from a telephone and form an appropriate Braille character by activating one or more solenoids. It can also send data back over the phone line. The current model has six keys that are arranged to match the pattern of keys on a standard Brailler. In addition, Doctor K intends to add a typewriter keyboard to the unit, making the transmission of messages simpler and faster. The keyboard unit causes as many problems as it solves, however, since it is far easier to send a message quickly than to receive it.

When a keyboard is added to the Telebraille, incoming data will have to be stored so that it can be read, one character at a time, at a comfortable pace by a blind person. Such a buffered Telebraille will also enable the deaf-blind to work with computers, because the data sent and received by

The microcomputer represents a terrific way to build new machines for the deaf-blind.

ROMtutorial ROMtutorial

Five-level Baudot code: An information transmission code used for Teletype transmission. Each character is defined by a sequence of five on-or-off bits; therefore only thirty-two distinct characters are possible.

Seven-level ASCII code: A standard code used to represent letters and numbers in binary on-or-off form. Seven distinct binary units, or bits, are used, allowing 128 different combinations.

Hertz: A unit of frequency equal to one cycle per second.

Buffer: A "machine" designed to be inserted between other machines or program elements to match impedances or peripheral equipment speeds, to prevent mixed interactions, to supply additional drive or relay capability, or simply to delay the rate of information flow.

ROMtutorial ROMtutorial

Interrupt: Relates to the suspension of normal operations or programming routines of microprocessors. Most often designed to handle sudden requests for service or change.

Duplex transmission: A simultaneous two way and independent transmission in both directions.

Acoustic coupler: A means of connecting a modem to a telephone through the handset, so that no electrical connection is needed. A sort of mechanical ear and mouth.

CPU: Central processing unit. The "thinking" portion of a computer. It decides where to move information, what to do to it when it's there, and where it should look for its next instructions.

BASIC: Short for Beginner's All-purpose Symbolic Instruction Code; the most common high-level language used by computer hobbyists.

Piezoelectric: A term applied to the phenomenon whereby certain crystalline materials develop useful electrical pressures (voltages) when the material is subjected to variable mechanical pressures, strains, or stresses; conversely, the materials develop mechanical strains or stresses when electrical voltages are applied.

the machine is in the standard ASCII format. Right now, Doctor K is using software to slow down data transmission from a computer to an unbuffered Telebraille, but he hopes to get to the next stage very soon.

In addition to a keyboard and a Braille cell readout, the Telebraille has an interrupt feature; the surface of the Telebraille can be made to vibrate. Each machine has an INTERRUPT button that can send a special signal to another Telebraille, even if the other Telebraille is sending data to the interrupter. Without that feature, which requires full duplex transmission, it would not be possible to get the attention of a deaf-blind person who is transmitting a message.

The interrupt feature lets you request a retransmission, interrupt a conversation that is taking a long time to go nowhere, send an emergency message, or, in the case of a Telebraille that is hooked to a computer, inform the Telebraille user of an imminent crash.

Another control on the Telebraille is a speed knob. It allows a user to pace the Braille characters. Depending on the setting of the speed control, a character may be held by the cell for a very brief moment or for several seconds. With a buffered Telebraille, the setting of the speed control will not affect the rate at which the sender can type a message.

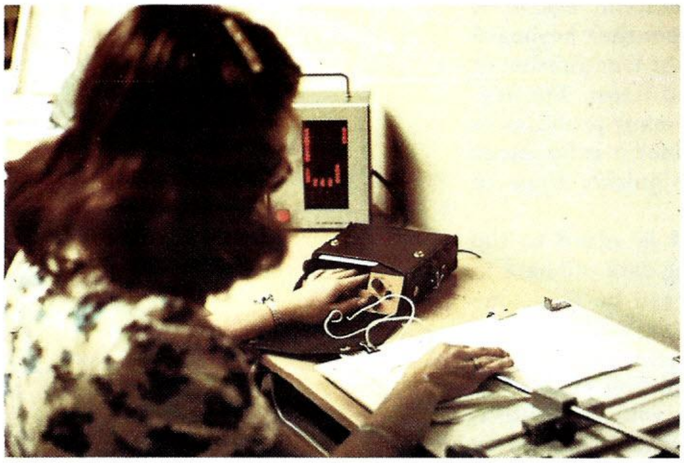
Doctor K's Telebraille is about the size of an attache case. There is no need to miniaturize it, even if the electronics can be made small, because the most important dimensions are determined by the placement of the Braille cell, keyboards, controls, and acoustic coupler.

The Telebraille will ultimately be controlled by a microcomputer. Doctor K favors the SC/MP, because it is suited to the table look-up operations that are used to translate between codes. By the time the machine is in production, however, he may well have chosen another type of computer. Whatever the CPU, Doctor K will use the Helen Keller Center's Interdata 70 minicomputer to assemble the software; other programs to test the Telebraille will be written on the same machine. Doctor K writes his software in BASIC so others may use it on whatever type of computer they have.

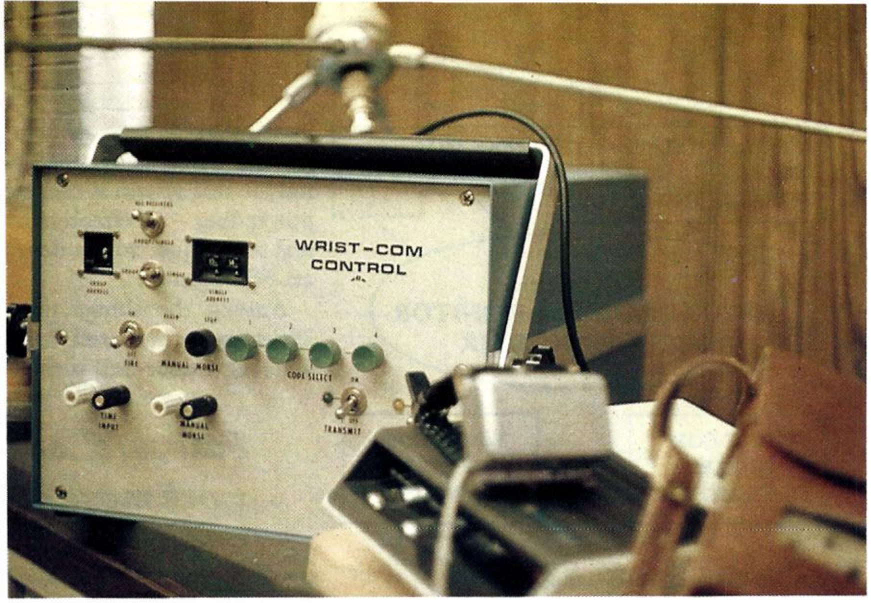
If you want to help Doctor K develop and build Telebraille machines, you should get in touch with him. The address is Helen Keller Center, 111 Middle Neck Road, Sands Point, N.Y. 11050. He needs volunteers to build and test the units, and in return he will teach you how to build machines that are safe and reliable enough to be used by the deaf-blind. An esteemed fellow of the Tom Sawyer school of motivational psychology, the amazing Doctor K has assured me that the work in his lab is great fun.

If you want to build your own Telebraille, you can do it for about $450—plus, of course, your time. Again, Doctor K will help you if you write him for information. He suggests that you think of visiting him and working in his shop before you try to build your own machine.

Another device you might want to make if you can figure out how to build a Telebraille is a Touchtone-to-Braille receiver. It could be used to enable a person who can hear to converse with a deaf-blind person who can



Opticon, an electronic device containing a photo diode matrix camera [at right], scans a line of print at a time. The image formed on the 144-diode matrix is translated into tactile points for touch-reading by the blind or deaf-blind [center]; at the same time the information is translated into a visual image for sighted persons [left]. The device, used here by Ms. Michelle Smithdas at the National Center, is being modified for speech output by its manufacturer, Telesensory Systems, Inc., in Palo Alto.



The Wrist-Com, soon to be a two-way wrist radio for the deaf-blind, allows one-character Morse and vibratory time, fire alarm, and other such survival signals to be sent to any wearer of a Wrist-Com receiver. The contents of the leather case—Motorola transmitter and receiver logic—and the silver can will be reduced in the course of development at the National Center to a unit wearable on the wrist. Present plans are to replace most of the discrete logic in the receiver with a microprocessor such as RCA COSMAC 1802.

speak (of which there are many). And a small unit that will produce vibrations in response to Touchtones would enable Morse code to be sent to a deaf-blind person by telephone.

Doctor K's most complicated project is the Institutional Wrist-com, which is being developed with the assistance of researchers at Stanford Research Institute under a contract with NASA's Technology Utilization Office.

"It is," he explains, "a wireless communications device with a receiver worn on the wrist. It also has a small transmitter to permit the deaf-blind user to acknowledge receipt of a message. In other words, it's a two-way wrist radio for the deaf-blind."

The Institutional Wrist-com allows a base station operator to address all persons at the site where it is installed (the Helen Keller Center in Doctor K's case). It provides a way to reach a group of people or an individual, too. The Institutional Wrist-com transmits both survival information and information of general utility, such as the time of day.

As he explained the idea to me, Doctor K picked up a handful of electronic equipment and grabbed my arm. He wrapped a Velcro band around my wrist. On it was a small silver box. Then he went over to a console and began flipping switches. As he was working, he handed me a small black and silver box—a modified radio receiver like those used to page people to the telephone—and told me to put it in my hip pocket.

"I'll get to that thing later."

Finally, he put a shoulder bag on my arm.

"This whole set-up is a prototype. It's crude, but it works."

The silver can on my wrist contained a tiny radio receiver. When Doctor K pushed a button on the control console, it made my wrist shake. Inside, a small electric motor with an eccentric weight was spinning away.

"That's to get your attention. Now put your finger over that little dot on top of the receiver."

I felt the spot buzz in response to Doctor K's activities at the console.

"It's Morse code," grinned the doctor, twiddling away at the controls. "Once I've gotten your attention with the first vibration, I can send you a message with the buzzing spot. It's piezoelectric, and it uses up less battery power than the motor."

"Each unit has a unique address. A common frequency reaches all the Wrist-com wearers in a given area. But each is assigned a digital access code unique to a particular receiver. That's how we can address only those persons we want to reach at any time."

I now had both a transmitter and a receiver swapping signals with the console, via my arm. Doctor K hit the controls again, and again I felt the vibration in my wrist, then a pause, then another vibration.

"Press that little button on the side of the wrist unit."

I did, and the vibrations ceased. The Wrist-com gave me three chances to acknowledge the page. If I had not responded, the control unit would have informed the console operator. I might have been a deaf-blind person in trouble.

If I hadn't responded, the control unit would have informed the console operator. I might have been a deaf-blind person in trouble.

The tiny receiver had no antenna of its own. Inside the Velcro wrist strap were two pieces of metal foil that coupled the radio to my arm. I was the antenna.

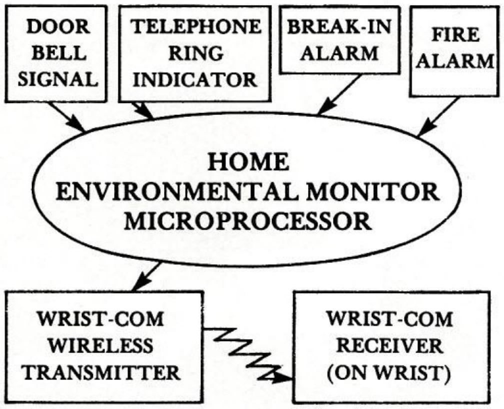
"The impedence of your arm is about fifty ohms," said Doctor K. That turns out to be a relatively easy impedence to match.

Doctor K plugged the shoulder bag into the receiver. He apologized for the bulk of the unit, and promised he would try to put the whole rig into a can the size of the receiver on my wrist just as soon as he could get miniaturized circuitry.

To process the calls and acknowledgements, Doctor K expects the Wrist-com will need a computer. The COSMAC processor, in his opinion, is well-suited to the present design.

Suddenly the receiver I had put in my hip pocket, and by now forgotten about, began to shake. As I reached for it to see what I might have to do to quiet it down, Doctor K spoke: "It's a time signal. It must be ten to the hour."

My watch agreed. The time signal had been broadcast over a radio fre-


DOOR BELL SIGNAL
TELEPHONE RING INDICATOR
BREAK-IN ALARM
FIRE ALARM
HOME ENVIRONMENTAL MONITOR MICROPROCESSOR
WRIST-COM WIRELESS TRANSMITTER
WRIST-COM RECEIVER (ON WRIST)

1. Coded doorbell signals possible
2. Telephone—cyclic on/off vibration
3. — · · · Break-in
4. · · — · Fire

WRIST-COM SIGNAL SYSTEM

quency which had been assigned to this project by NASA. The transmitter and pocket receiver I held would eventually be replaced by the Institutional Wrist-com. It might also evolve into a unit that is part of a community-wide signaling service, but that kind of system for the deaf-blind is still part of some hoped-for future.

A near cousin of the institutional model, the Residential Wrist-com, has been designed for use in the home or apartment by one or two deaf-blind persons living together. The radio signal for this system is radiated from the power line—an effective and economical technique, which does not require an FCC license.

Among the things the Residential Wrist-com could tell its wearer are: whether a door-bell is ringing, if there is a door open or a fire alarm, whether it is raining outside, how something on the stove is doing, and whether or not the telephone is ringing. Each Residential Wrist-com will have a microcomputer and a complex of sensors. By sending attention signals and Morse code, the wearer will be able to cope with a house or apartment. A Morse code F could mean fire, a T could mean telephone. By asking friends to ring the doorbell in a special pattern, the deaf-blind person could distinguish the familiar from the unfamiliar. And if the Residential Wrist-com required acknowledgement, it could be programmed to call others (or other computers) in the event of an emergency. Perhaps a smart Wrist-com would not only tell the wearer about a fire, but also suggest the best exit. The possibilities are endless, and many would not radically increase the cost of the system. One of Doctor K's jobs is to figure out the best balance between cost and utility.

Like every survival device for the deaf-blind, the Telebraille and the Wrist-com must be self-testing. It is difficult for a deaf-blind person to know whether equipment is working properly without some special feature. All such fail-safe engineering makes Doctor K's work more difficult—and more interesting.

Fail-safe engineering makes Doctor K's work more difficult—and more interesting.

The wonderful dreams of Doctor K will all come true. Some are almost realized already, and all of them are being tested by the Helen Keller Center's clients at some stage of development. But even when the Wrist-com and Telebraille are completed, they will be the appliances of only the most fortunate deaf-blind people. There are thousands who may never be made aware of Doctor K's work and of the hopes of all the people at the Center. For most deaf-blind people, the world is not only silent and dark, it is indifferent.

"We've had people come here who don't even know their names," laments Dr. Frederick M. Kruger. "As a matter of fact, they didn't even know that they had names! Some have been locked in an institution for thirty years because they were 'retarded'—meaning that nobody could communicate with them. Nobody even realized they couldn't be reached because they were deaf and blind."

That's one of the thoughts that keeps Doctor K running. ▼

FROM INKPRINT TO BRAILLE

Translating inkprint to Braille is a lot more than simply substituting one symbol for another. Because Braille is not easy to read fast, books, articles, and the like (but not computer programs or foreign language text) are produced in a condensed form. It's sort of like speedwriting. Braille that is fully spelled out is called Grade I; Braille that is condensed is called Grade II.

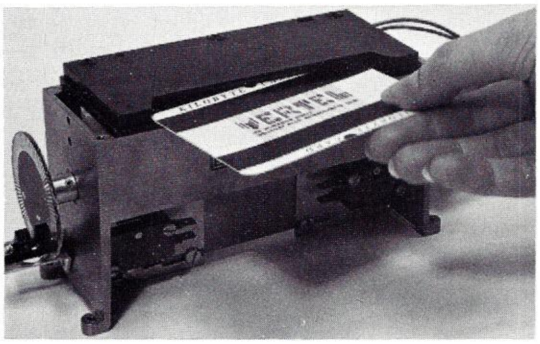
The rules for condensation are very complex. As a result, most inkprint-to-Braille translation is done by people. There is a large COBOL program at M.I.T. that produces Grade II Braille, called DOTSYS III. It works with a Braille page printer called the Braillemboss.

A similar program that would run on a home computer would be of enormous benefit. To develop it, you will need a configuration that includes some storage, as well as some kind of embossing device. However, you could begin by writing a program that puts out Grade II Braille on any terminal, switching to an embosser at a later date.

The standard guide to condensation, *Transcribers' Guide to English Braille,* by Bernard M. Krebs, is available from the Jewish Guild for the Blind, 15 West 65 Street, New York, NY 10023. The inkprint edition costs $3.00 postpaid and the Braille edition costs $12.80 postpaid. With the *Guide,* all you need is a computer and you're on your way.

The Helen Keller National Center was established by unanimous Act of Congress in 1969 and is funded through the Department of Health, Education, and Welfare. Doctor Kruger is actually only one of almost 100 dedicated people working at the National Center with the common goal of improving and expanding life for deaf-blind persons.

Run On Micro



KILOBYTE CARD Memory for Pennies

by Thorn Veblen

"What's on the flip side of *Star Trek?*"

"Explorer II. But my best card's the *Star Wars* double from Gamex with the *Wookie Waltz* back to back."

"Sure, sure. But I'll tell you a real buy. Jennie's Discount's selling their astronomy series for 99 cents this week."

Kids trading bubble gum cards? A new jargon for records? Neither of these, yet in a way both—for personal computing is about to take a giant step forward into mass marketing to Everyman.

The first brick has been laid in the void of software. A towering library of inexpensive, readily available programs is only a year or two away for the home computer user. What prompts this optimism about finding a path through the software jungle is the development of the Kilobyte Card by Vertel.

Though as with any mass market information storage media—the 45 record versus the 33, tape cassettes versus cartridges, etc.—the final parameters of the dominant product are hard to determine, depending not only on consumer acceptance but marketing and packaging skills as well. But right now there's little doubt that it will be some form of Kilobyte Card and its operating companion will probably be some form of the KB-31 Microloader.

The Kilobyte Card is basically the same tried and true magnetic strip credit card held so dearly by tens of millions of American wallets. Certainly no one will think of it as a new gimmick, which means the hurdle of consumer acceptance has been passed before the obstacle even appeared.

The difference between this memory credit card and a regular card for the average consumer is primarily one of stripes. The Kilobyte Card has four stripes rather than one. The difference for the curious is that each stripe or track has two channels which are read or recorded simultaneously but independently. Also unlike the standard magnetic strip encoded card, which stores only 119 four-bit characters—really an inadequate amount of memory for any applications program—the Kilobyte Card carries a full 1,024 eight-bit words. Even greater density is expected in the near future, with a 4K card probably hitting the racks by the end of '78. And by racks I mean rotating carousels of gaudily blister-packed program cards to choose from at your local computer store just the way you choose paperback books—and at a price not much higher. At, say, $3.98 for a single and $5.98 for a double, a "best seller" could probably sell 10-25,000 copies a year by 1979, with sales of singles running into the millions by the 1980s. Since the cards can be cranked out for a dime apiece, there's plenty of profit left for the manufacturer and retailer. More importantly, it also can mean a good royalty for the writer of the software. The cards will be so inexpensive it won't pay for any users but the diehard crook to rip off the programs.

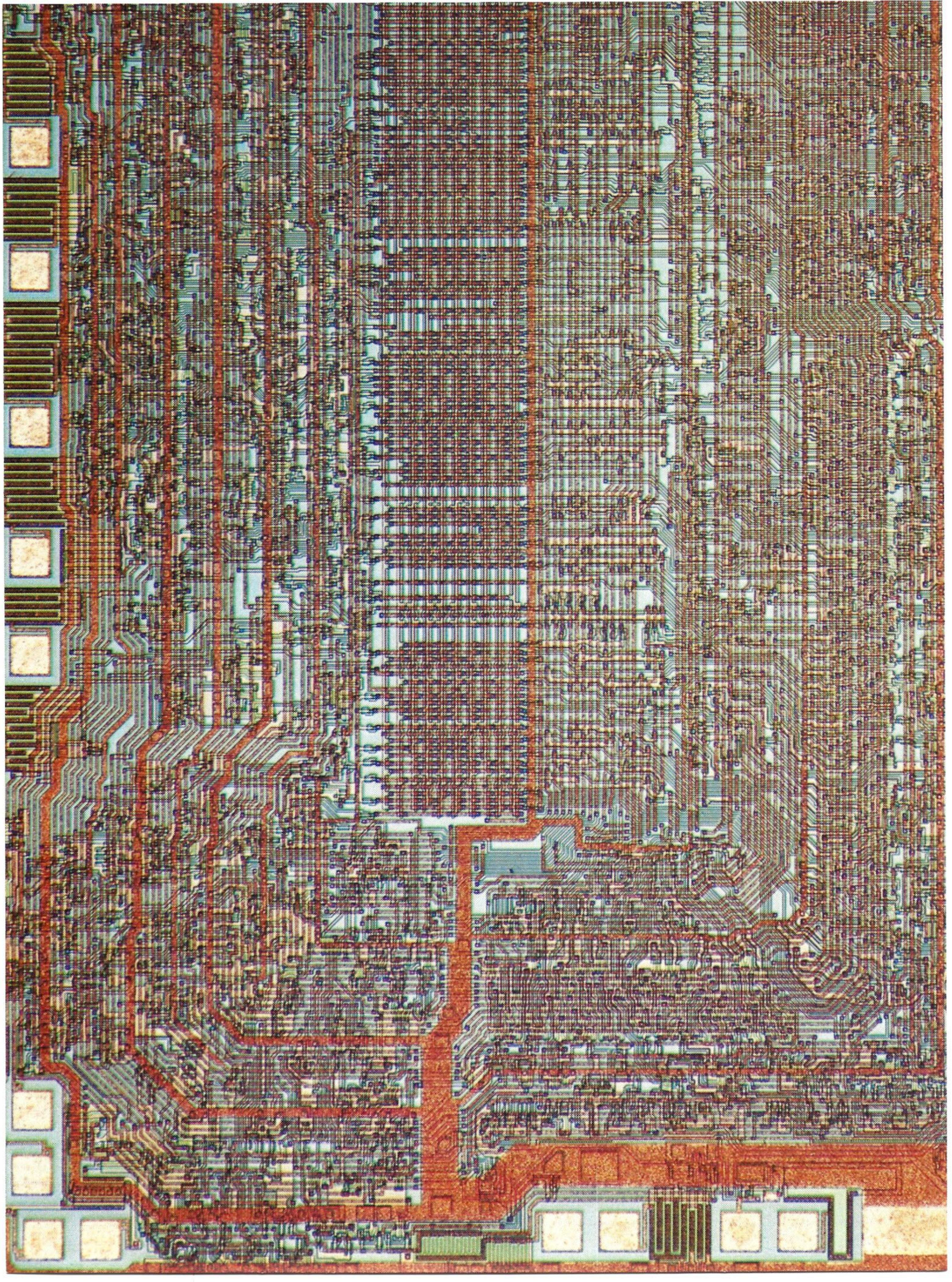
The Kilobyte Card's main drawback is the slowness of access. With a reading speed of only five kilobytes per minute, it's not something one would normally associate with computers, even the relatively slow micros. But that's if you're a technician. From the average user's point of view it represents no more of a hinderance than the icebox raid during prime-time commercials.

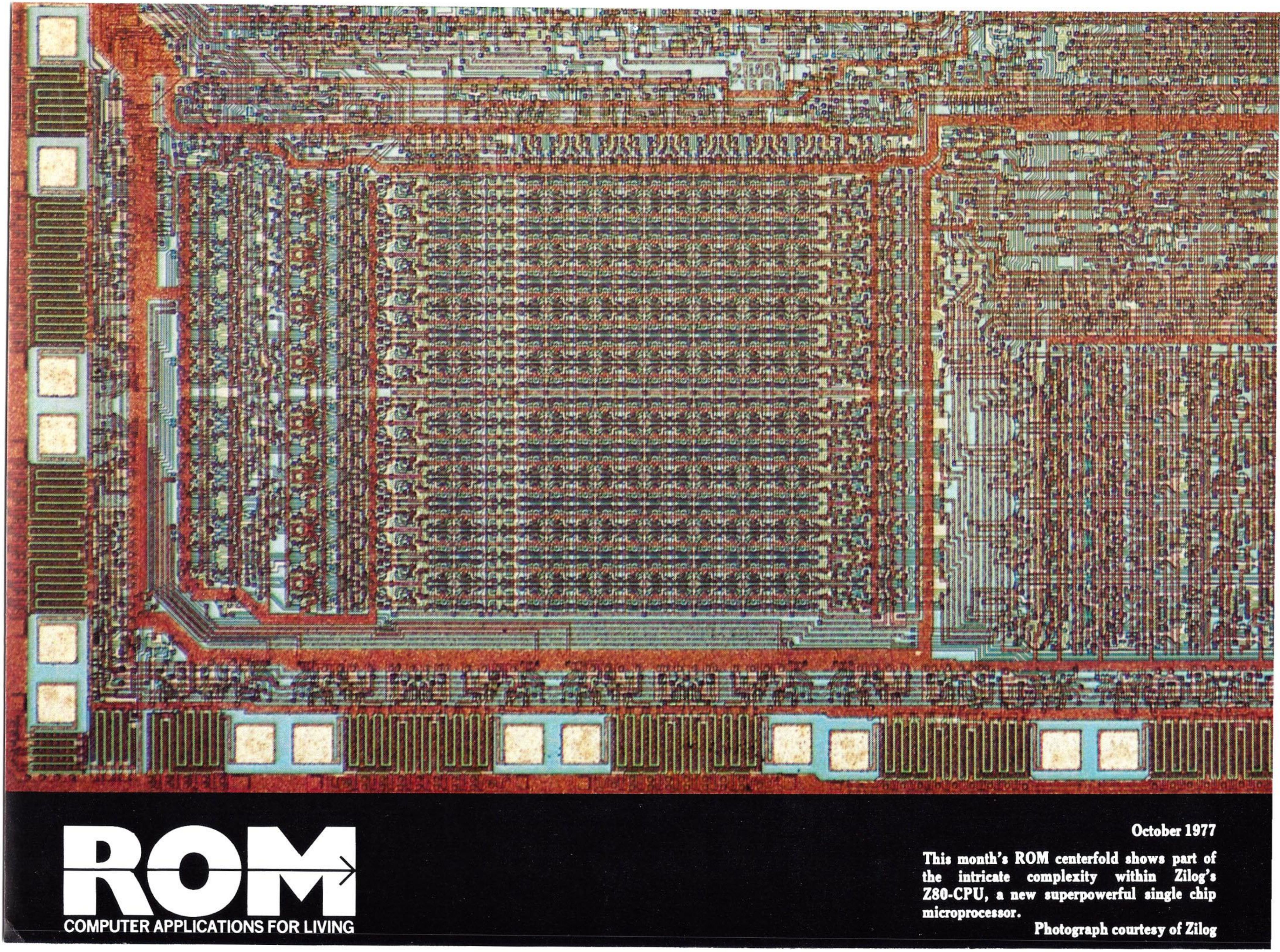
The clincher is that, unlike floppies, cassettes, and calculator data strips, these cards can handle the type of abuse the *enfant terrible* of the technological world—the average consumer—can expect to heap on them without damage. Ever try to wipe Koolaid and peanut butter off a floppy? No problem with a Kilobyte Card.

The best of memories are, of course, useless without a reader, or, better yet, a reader/writer. Well, the companion KB-31 reader/writer by Vertel accidentally meets the most stringent of consumer demands: It can survive abuse by the barrelful. This accidental sturdiness came about because it was originally designed for operation outdoors as well as in hostile factory environments, making it about as rugged as computer peripherals can get.

Although the price as yet is not quite right, with the reader/writer selling for $99 in quantities of 1000 and another $159 for the electronics, including a fully buffered +12V RS-232 output, it could be retailed within the turntable/hi-fi range. And if the demand reaches its potential, it would be no great feat to stick the electronics on a chip, substitute the precision metal castings with molded plastics and stick the whole affair in an attractive teak or oak cabinet. Presto, there's the "designer styled" home-applications minimemory unit for $59.95—with data loading no more complex than dropping a couple of slices of bread into the toaster.

Which brings up the last plus. Designed with even the most unskilled operator in mind, the KB-31 will not function unless the card is inserted properly. Someone would really have to work at it to screw things up. This feature permits the honest application of that invaluable old advertising line, "So simple any child can use it." In other words, the home computer system is beginning to come of age. ▼





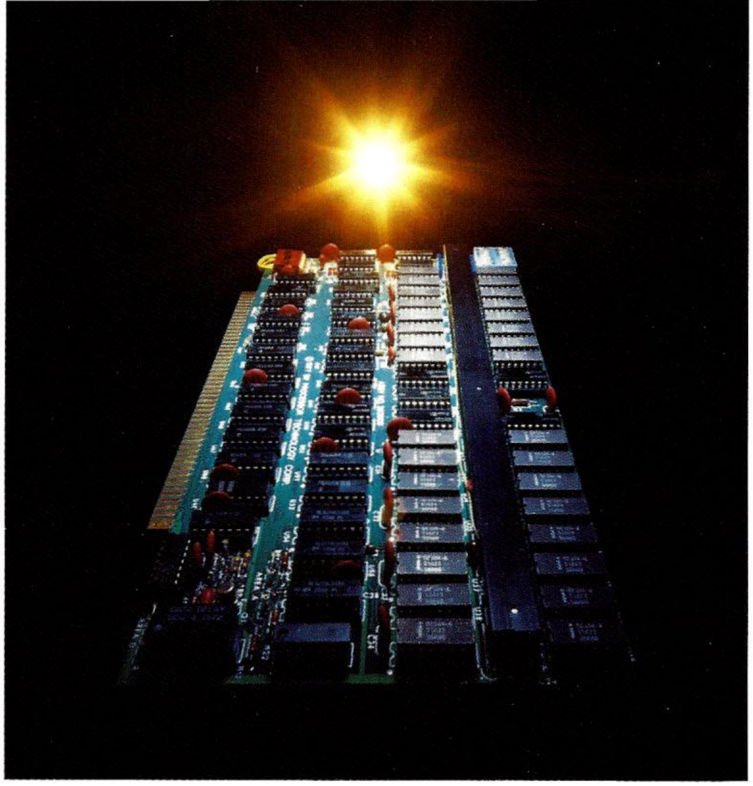
ROM

COMPUTER APPLICATIONS FOR LIVING

October 1977

This month's ROM centerfold shows part of the intricate complexity within Zilog's Z80-CPU, a new superpowerful single chip microprocessor.

Photograph courtesy of Zilog



The end of Kit-Kits.

The end of bad solder joints, heat damaged components and sick IC's. Introducing the Semikit. Item 1, a 16KRA Memory Board, $369.

Let's face it. Loading and soldering PC Boards is not much fun for the kit builder. Even more important, it's the place where most of the trouble gets introduced. The real fun and education comes in running and testing boards.

Now the Semikit with fully tested IC's.

At the price of a kit, Processor Technology Corporation introduces the Semikit. It's a fully stuffed, assembled and wave soldered PC Board loaded with IC's that have gone through Q.C. and final check-out (a first in the industry).

We leave you the fun of testing with our fully documented set of instructions. We do the production tasks of loading, wave soldering and inspecting the boards. You do the more interesting and time consuming chore of testing and burning-in the boards.

The result is one sweet deal for both of us. You get a board where the primary causes of damage (poor solder joints, excess solder and bad IC's) are virtually eliminated. You get a board of highest professional quality. And we get the business!

The 16KRA Memory Board's at your dealer now.

Your Processor Technology dealer has the first Semikit, a 16KRA Memory Board, in stock and ready to go right now. You can take it home tonight for $369 as a Semikit or for $399 fully assembled, tested and burned-in.

You'll have a 16,384 byte memory with a better price performance ratio than anything on the market today. Now you can afford to add quality, high density memory to your system for remarkably little. And you can add enough to solve complex computing problems right in the main frame.

The memory features invisible refresh. There's no waiting while the CPU is running. Worst case access time is 400 nsec. Each 4,096 word block is independently addressable for maximum system flexibility. Power is typically 5 watts, the same as most single 4K memory modules. Back-up power connection is built-in.

Other Semi's are coming your way.

The 16KRA Memory is Processor's first step in adding more fun, capability and reliability to your computer system at lower cost. Other modules are on the way to your dealer now. Come on down today.

Or you may contact us directly. Please address Processor Technology Corporation, Box O, 7100 Johnson Industrial Drive, Pleasanton, California 94566. Phone (415) 829-2600.

Processor Technology

See Sol here...

ALABAMA

ICP, Computerland
1550 Montgomery Hwy.
Birmingham, AL 35226
(205) 979-0707

ARIZONA

Byte Shop Tempe
813 N. Scottsdale Rd.
Tempe, AZ 85281
(602) 894-1129

Byte Shop Phoenix
12654 N. 28th Dr.
Phoenix, AZ 85029
(602) 942-7300

Byte Shop Tucson
2612 E. Broadway
Tucson, AZ 85716
(602) 327-4579

CALIFORNIA

The Byte Shop
1514 University Ave.
Berkeley, CA 94703
(415) 845-6366

Computer Center
1913 Harbor Blvd.
Costa Mesa, CA 92627
(714) 646-0221

DCI Computer Systems
4670 N. El Capitan
Fresno, CA 93711
(209) 266-9566

Bits 'N Bytes
679 S. State College Blvd.
Fullerton, CA 92631
(714) 879-8386

The Byte Shop
16508 Hawthorne Blvd.
Lawndale, CA 90260
(213) 371-2421

Opamp/Computer
1033 N. Sycamore Ave.
Los Angeles, CA 90038
(213) 934-3566

The Computer Mart
624 West Katella #10
Orange, CA 92667
(714) 633-1222

Byte Shop
496 South Lake Ave.
Pasadena, CA 91101
(213) 684-3311

Micro-Computer
Application Systems
2322 Capitol Avenue
Sacramento, CA 95816
(916) 443-4944

The Computer Store
of San Francisco
1093 Mission Street
San Francisco, CA 94103
(415) 431-0640

Byte Shop
321 Pacific Ave.
San Francisco, CA 94111
(415) 421-8686

The Byte Shop
2626 Union Avenue
San Jose, CA 95124
(408) 377-4685

The Computer Room
124H Blossom Hill Rd.
San Jose, CA 95123
(408) 226-8383

The Byte Shop
509 Francisco Blvd.
San Rafael, CA 94901
(415) 457-9311

The Byte Shop
3400 El Camino Real
Santa Clara, CA 95051
(408) 249-4221

Recreational Computer
Centers
1324 South Mary Ave.
Sunnyvale, CA 94087
(408) 735-7480

Computer Components
5848 Sepulveda Blvd.
Van Nuys, CA 91411
(213) 786-7411

The Byte Shop
2989 North Main St.
Walnut Creek, CA 94596
(415) 933-6252

Byte Shop
14300 Beach Blvd.
Westminster, CA 92683
(714) 894-9131

COLORADO

Byte Shop
2040 30th St.
Boulder, CO 80301
(303) 449-6233

Byte Shop
3464 S. Acoma St.
Englewood, CO 80110
(303) 761-6232

FLORIDA

Byte Shop of Miami
7825 Bird Road
Miami, FL 33155
(305) 264-2983

Microcomputer
Systems Inc.
144 So. Dale Mabry Hwy.
Tampa, FL 33609
(813) 879-4301

GEORGIA

Atlanta Computer Mart
5091-B Buford Hwy.
Atlanta, GA 30340
(404) 455-0647

ILLINOIS

Champaign Computer
Company
318 N. Neil Street
Champaign, IL 61820
(217) 359-5883

itty bitty machine co.
1316 Chicago Ave.
Evanston, IL 60201
(312) 328-6800

itty bitty machine co.
42 West Roosevelt
Lombard, IL 60148
(312) 620-5808

INDIANA

The Data Domain
406 So. College Ave.
Bloomington, IN 47401
(812) 334-3607

The Byte Shop
5947 East 82nd St.
Indianapolis, IN 46250
(317) 842-2983

Computers Unlimited
7724 East 89th Street
Indianapolis, IN 46256
(317) 849-6505

The Data Domain
7027 N. Michigan Rd.
Indianapolis, IN 46268
(317) 251-3139

IOWA

The Computer Store
of Davenport
616 West 35th Street
Davenport, IA 52806
(319) 386-3334

KENTUCKY

The Data Domain
3028 Hunsinger Lane
Louisville, KY 40220
(502) 456-5242

MICHIGAN

The Computer Store
of Ann Arbor
310 East Washington
Ann Arbor, MI 48104
(313) 995-7616

Computer Mart
of Royal Oak
1800 W. 14 Mile Rd.
Royal Oak, MI 48073
(313) 576-0900

General Computer Store
2011 Livernois
Troy, MI 48084
(313) 362-0022

MINNESOTA

Computer Depot, Inc.
3515 W. 70th St.
Minneapolis, MN 55435
(612) 927-5601

NEW JERSEY

Hoboken Computer Works
No. 20 Hudson Place
Hoboken, NJ 07030
(201) 420-1644

The Computer Mart
of New Jersey
501 Route 27
Iselin, NJ 08830
(201) 283-0600

NEW YORK

The Computer Mart
of Long Island
2072 Front Street
East Meadow, L.I. NY 11554
(516) 794-0510

The Computer Shoppe
444 Middle Country Rd.
Middle Island, NY 11953
(516) 732-4446

The Computer Mart
of New York
118 Madison Ave.
New York, NY 10001
(212) 686-7923

The Computer Corner
200 Hamilton Ave.
White Plains, NY 10601
(914) 949-3282

OHIO

Computer Mart of Dayton
2665 S. Dixie Ave.
Dayton, OH 45409
(513) 296-1248

OREGON

Byte Shop Computer Store
3482 SW Cedar Hills Blvd.
Beaverton, OR 97005
(503) 644-2686

The Real Oregon
Computer Co.
205 West 10th Ave.
Eugene, OR 97401
(503) 484-1040

Byte Shop Computer Store
2033 SW 4th Ave.
Portland, OR 97201
(503) 223-3496

PENNSYLVANIA

Byte Shop of
Delaware Valley
1045 Lancaster Pike
Bryn Mawr, PA 19010
(215) 525-7712

RHODE ISLAND

Computer Power, Inc.
M24 Airport Mall
1800 Post Rd.
Warwick, RI 02886
(401) 738-4477

TEXAS

Computer World
926 N. Collins
Arlington, TX 76011
(817) 469-1502

Byte Shop
3211 Fondren
Houston, TX 77063
(713) 977-0664

Computertex
2300 Richmond Ave.
Houston, TX 77006
(713) 526-3456

Interactive Computers
7646½ Dashwood Rd.
Houston, TX 77036
(713) 772-5257

Neighborhood Computer
Store
#20 Terrace Shopping Center
4902 - 34th Street
Lubbock, TX 79410
(806) 743-2787

The Micro Store
634 So. Central
Expressway
Richardson, TX 75080
(214) 231-1096

VIRGINIA

The Computer Systems
Store
1984 Chain Bridge Rd.
McLean, VA 22101
(703) 821-8333

Media Reactions Inc.
11303 South Shore Dr.
Reston, VA 22090
(703) 471-9330

The Home Computer Center
2927 Virginia Beach Blvd.
Virginia Beach, VA 23452
(804) 340-1977

WASHINGTON

Byte Shop Computer Store
14701 N.E. 20th Ave.
Bellevue, WA 98007
(206) 746-0651

The Retail Computer Store
410 N.E. 72nd
Seattle, WA 98115
(206) 524-4101

WISCONSIN

Madison Computer Store
1910 Monroe St.
Madison, WI 53711
(608) 255-5552

The Milwaukee
Computer Store
6916 W. North Ave.
Milwaukee, WI 53213
(414) 259-9140

CANADA

Trintronics
160 Elgin St.
Place Bell Canada
Ottawa, Ontario K2P 2C4
(613) 236-7767

First Canadian
Computer Store Ltd.
44 Eglinton Ave. West
Toronto, Ontario M4R 1A1
(416) 482-8080

The Computer Place
186 Queen St. West
Toronto, Ontario M5V 1Z1
(416) 598-0262

Pacific Computer Store
4509-11 Rupert St.
Vancouver, B.C. V5R 2J4
(604) 438-3282

Processor Technology

PCE PERSONAL COMPUTING EXPO
NEW YORK COLISEUM, OCTOBER 28, 29, 30, 1977

General Information

You may find it advantageous to purchase two or three-day admission tickets in advance. These are available by mail only, no later than October 10, 1977. Use coupon below.

Group rates (10 or more persons) qualify for $1.00 off regular prices. Arrangements must be made by mail prior to October 10, 1977.

Special arrangements have been made if you desire to stay overnight. Our headquarters hotel, the Barbizon-Plaza, is located on Central Park South, two blocks from Columbus Circle. Single rooms available at $34.00 per night; $40.00 double, plus tax. There's a weekend plan: $22.95 daily, plus tax per person, double occupancy . . . includes breakfast (brunch on Sunday) and meal gratuities. Children under 14 in same room with parents, free.

For hotel reservations and information, call toll free (800) 223-5493. From New York State call (800) 223-5963.

For those traveling to New York by air, American Airlines offers a convenient service through arrangement with Personal Computing Expo. For information, call toll free (800) 433-1790. In Texas the number is (800) 792-1150. From the West Coast, round trip fare via American is only $227.00.

20,000 persons are expected to attend and view the more than 200 exhibits by personal computer manufacturers and retailers.

Personal Computing Expo will occupy the 4th floor of the New York Coliseum. It is located on 59th Street and Columbus Circle — the geographical center of New York City. Garage parking in the building is available.

For answers to any questions pertaining to your attendance at Personal Computing Expo, contact the Show Manager, Ralph Ianuzzi, at Area Code 212/753-4920.

Show Hours and Admission

Personal Computing Expo hours are as follows:
Friday, Oct. 28 — Noon to 10 p.m.
Sat. Oct. 29 — 10 a.m. to 10 p.m.
Sunday, Oct. 30 — Noon to 7 p.m.
General Admission: $5.00 (includes free BYTE lectures) per day.
Two-day Tickets: $9.00 (advance sale only)
Three-day tickets: $13.00 (advance sale only)

Exciting lectures sponsored by BYTE

Personal Computing Expo is endorsed by BYTE magazine, whose staff has contacted prominent speakers for an exciting series of lectures.
Visitors will be able to attend these meetings **free of charge.** The lectures will not conflict with each other eliminating the worrysome choice among several equally important topics. In addition, they will be repeated on the next day to give you a second chance if you missed a topic.
Lectures are typically 30 minutes, often with demonstrations and an additional 15 minutes for questions.

Louis E. Frenzel
Jack L. Davies
Carl Helmers
Sol Libes
John H. Dilks III
Portia Isaacson Ph.D.
David Fylstra
Max Mathews Ph.D.
Carl L. Holder
Robert S. Jones

Personal Computing Expo admission is $5.00 per day. Advance reservation eliminates waiting in line. Order advance tickets with this coupon. Admission ticket includes access to exhibits, lectures and tutorials.

Please send me ________________ advance registration tickets for three days, October 28-29-30. Total cost $13.00 per person.
Please send me ________________ advance tickets for two days, October ______ and October ______. Cost is $9.00 per person.
Please send me ________________ advance tickets for one day, October ______. Cost is $5.00 per person.
Make all checks payable to PERSONAL COMPUTING EXPO, and mail to:
Personal Computing Expo, 78 East 56th Street, New York, N.Y. 10022.
Name______________________________ Amount enclosed $____________
Address ____
City__________________________State________________Zip______________

The Unlikely Birth of a Computer Artist

by Richard Helmick

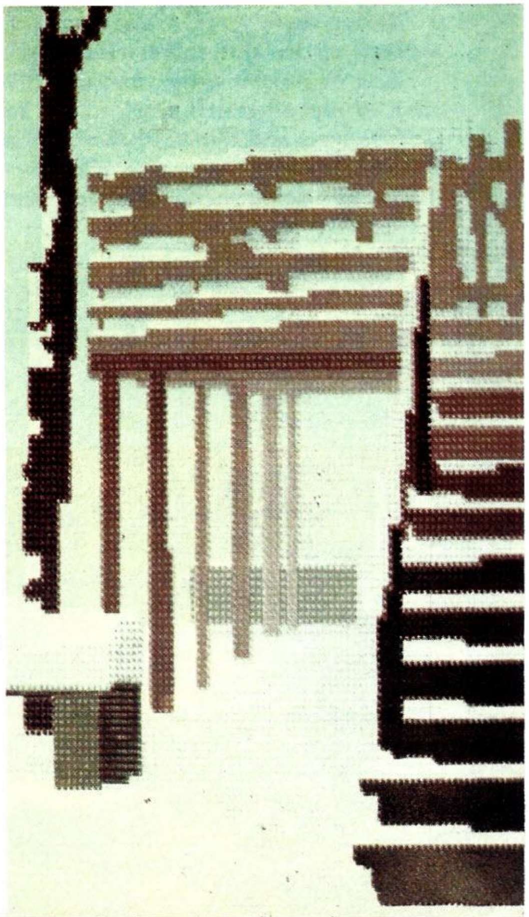
Not everyone looking at an IBM 370/168 computer would think at once of art, design, architecture, or film possibilities. I certainly didn't. However, in the early seventies, I turned to a computer for assistance with stereoscopic designs for use in the optical paintings I was doing at the time. Having no background in computer science, engineering, mathematics, or anything remotely related to computers, I sought assistance rather haphazardly.

By sheer coincidence, the biennial National Sculpture Conference, held at the University of Kansas in 1972, featured several sessions on computer graphics in the arts. The presentations of such computer art notables as Ken Knowlton and Jeff and Colette Bangert were so impressive that my original ideas concerning stereoscopic painting were pushed to the back burner in favor of a more general investigation of computer graphics in art and design. The plotter drawings and computer-animated films shown at the conference seduced me into computer art.

Furthermore, I happily discovered that people working in the art and technology interface are quite willing to share their knowledge with the uninitiated. The Bangerts and Ken Knowlton responded to an invitation to speak to me and my colleagues at the University of Missouri and subsequently shared graphic programs with me. I use these programs to this day.

Other conferences, such as the Symposium on Computer Art sponsored by the Cranbrook Academy of Art in

Deck



Photographs of the artist at work by Rhod Lauffer
Photographs of completed art by Thomas Hall

1973 and the Computer Aided Design Workshop held at Harvard University in 1974, were invaluable in helping me develop my current attitudes toward computer usage in the visual arts.

At about the same time, I discovered the Department of Electrical Engineering at the University of Missouri. The department has, among other things, a Spatial Data Systems Model 108 Computer Eye Scanner controlled by a Digital Equipment Corporation PDP-11/50 minicomputer. Photographs, drawings, etc. can be optically scanned. Levels or degrees of light and dark and the location of light and dark areas can be recorded numerically on magnetic tape. The recorded information can then be used by a computer program to cause a version of the original photograph or drawing to be printed out on an IBM 1403 high speed line printer controlled by an IBM 370/168 computer at the University of Missouri computer center. Since I had no other way to incorporate objective imagery into my work, I found this facility to be extremely valuable in the development of my art.

> *It can be a cultural shock for a person trained in the visual arts to truck with advanced technology.*

During these years, I also began to collect, trade, and collaborate in the writing of graphic programs. In addition to the programs given to me by Jeff Bangert and Ken Knowlton, I was involved in a collaborative effort between my department (Housing and Interior Design) and the Department of Computer Science, which yielded a program for drafting architectural floor plans. A graduate student in interior design, Judy Kleinsorge, and I worked out the features we wanted included in the program. Chris Korschgen, then a graduate student in computer science, wrote the program, called STRUCTURES, in PL1.

Basically, the user tells the program the length, angle, and location of walls, windows, doors, and other architectural features, fixtures, and furnishings. The floor plan is drafted on a Calcomp drum plotter. Once the program has the initial information, parts of the plan may be moved, enlarged, reduced, repeated, or added to in subsequent drawings by adding two or three appropriately punched cards to the input deck.

Unfortunately, in spite of our best efforts, getting information into the computer via punched cards turned out to be laborious at best. In addition, an off-line plotter tends to make turnaround unpredictable. I later traded STRUCTURES for a program capable of drawing architectural perspectives.

I learned some elementary FORTRAN through the use of a series of instructional video tapes developed by the University of Missouri at Rolla. However, the writing of programs remains difficult for me. Collaboration with a skilled programmer is still the best way for me to get programs written.

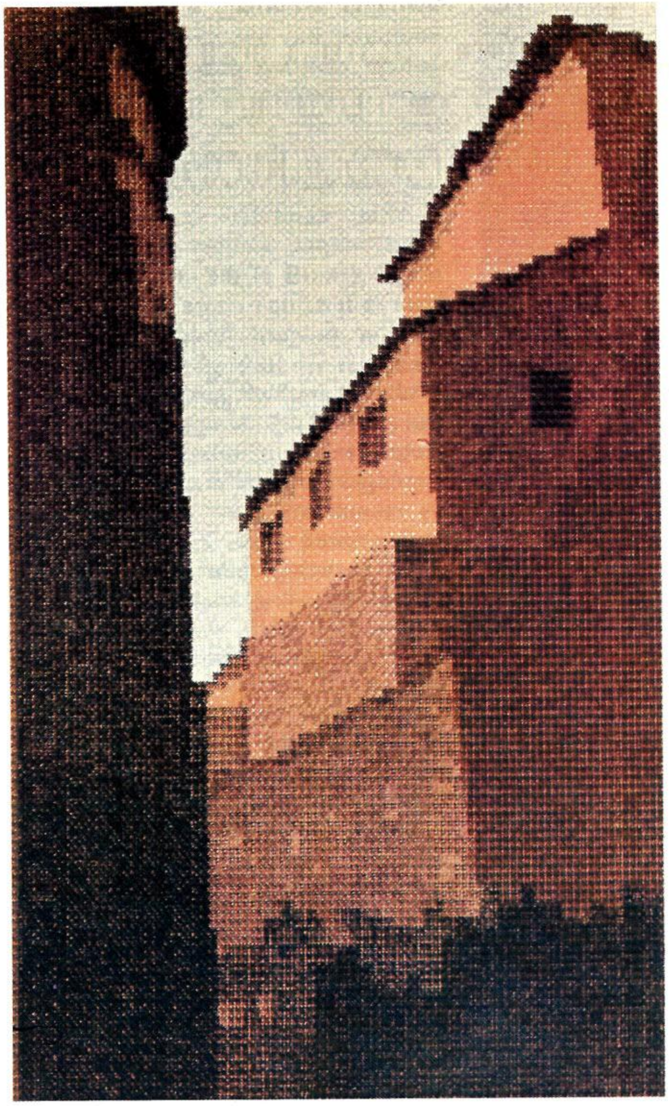
Based on the assumption that most students of art and art-related fields are, for some reason, culturally conditioned to fear advanced technology or are as ignorant as I was about computers a few years ago, I developed a course to expose students to the potential usefulness of computer graphics in art.

In the course, students are introduced to a variety of the programs I have begged, borrowed, traded for, or developed. They then use these programs in the design and execution of projects related to their individual interests and fields. Students of interior design might use computer graphics as an aid in the development and previsualization of interior components such as fabrics and rugs. Students of art might incorporate computer-generated drawings and patterns into printmaking. Students of architecture might use computer graphics to draft floor plans and perspectives.

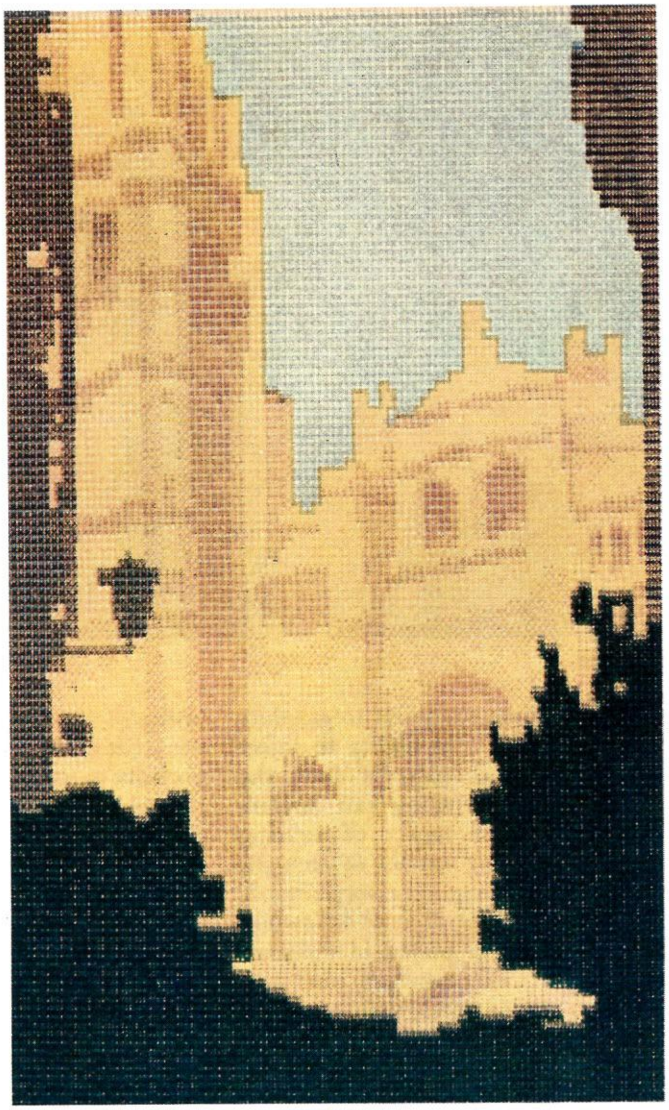
Even if a student goes no further than this introductory course, he has been exposed to the fact that computer graphics exist and can be useful in his field. If the course whets a student's appetite, he can then learn to program and continue to pursue his interest in computer graphics. A course of this nature seems to fill a gap between the concerns and skills of the artist and the concerns and skills of the technologist. But it can be a cultural shock for a person trained in the visual arts to truck with advanced technology.

In my own first attempts at generating computer-drawn graphics, I used packaged programs which were intended graphically to display quantitative data. I processed arbitrary data and came up with results which were unexpected, yet which suggested to me possibilities for aesthetic manipulation. By reversing positives and negatives, superimposing one computer pattern on top of another, using selected parts of patterns, and manipulating color, it is possible to create interesting works of art.

Serigraphy seemed to be an ideal medium for massaging selected computer patterns. While this idea was not original with me (I had seen screenprints by Ken Knowlton earlier), I felt



View of Toledo



Santo Tomé

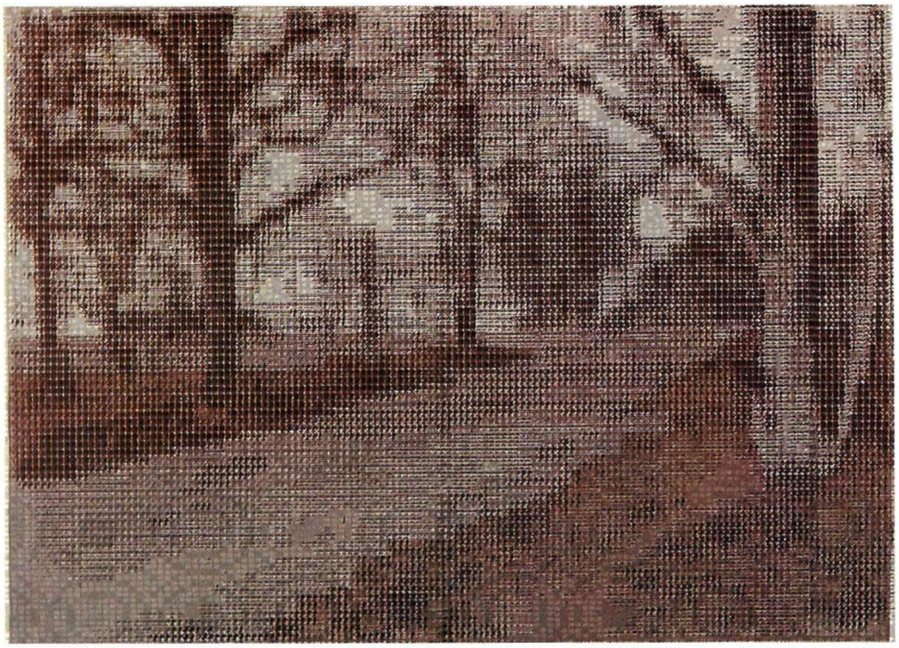
the particular patterns I would generate, and the individual ways in which I would manipulate them in the process of screening prints would, indeed, yield original works of art.

Basically, the process of translating computer graphic output into a screenprint begins by making line shot photographs of the computer-generated patterns on graphic arts film. The photographs are usually enlarged and include a positive as well as a negative line shot. Since I don't have access to the necessary photographic equipment, I hire the photographic work done by a local newspaper publisher.

Using these line shots, I make photo stencils in my studio. After stretching a multi-filament polyester fabric tightly on a wooden frame, I apply a coat of commercially prepared emulsion, made light-sensitive with ammonium bichromate, onto the stretched fabric. By placing one of the line shot photos on top of the stretched fabric, I can selectively expose the emulsion to light from number two photo floods. Light penetrates the emulsion through the clear portions of the photograph, but the emulsion is protected from the light under the opaque (black) portions. The emulsion hardens where exposed to light, but remains water-soluble where protected from light.

After an exposure of three to ten minutes, I wash out the soft, unexposed emulsion. Now the pattern on the stretched fabric (screen) is closed or plugged in the places where the original computer pattern was white and open where the original pattern was black. By placing a sheet of paper under the screen, I can force inks through the open places onto the paper. I can manipulate the computer patterns by printing one pattern and/or color on top of another, printing selected portions of the pattern only, or printing positive and/or negative versions of the pattern.

When beginning a work, especially those which contain no objective imagery, I have, at best, only a vague idea of the aesthetic goal I'm after. The initial computer designs I generate are rather random or, if not actually random, at least the results are unforeseen by me. It appears that one of the major con-



Westwinds

tributions of the computer to graphics as an art form is its ability to inject randomness and surprise into the work. Creative thinking can flourish only when some aspect of existing orders or existing modes of thought breaks down or is in some way altered. At the point of, or directly following, the breakdown, a period of randomness occurs. The random period is necessary to the emergence of a new order or a new mode of thinking. I don't believe the period of randomness guarantees the emergence of creative thoughts, but it acts, nonetheless, as a catalyst.

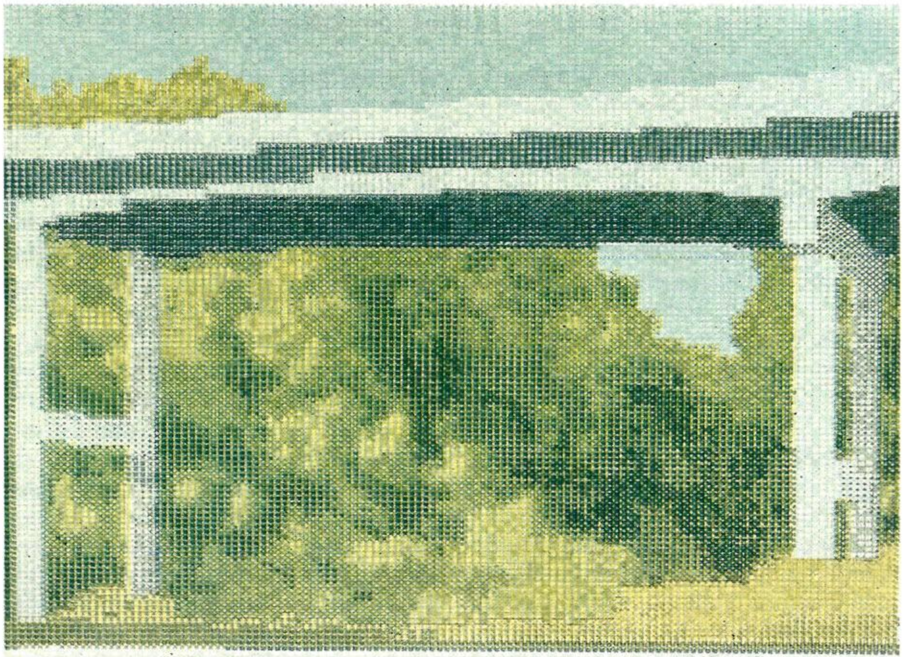
While this idea has become clearer and more forceful to me while I've been involved with computers, it is not a new thought. Artists through the ages have been aware of the value of the random and the unforeseen and have devised ways to introduce the accidental into their field of vision in order to induce creative thinking. Notable examples range from Leonardo's suggestion that looking at paint-spattered walls may aid invention to Jean Arp's practice of dropping cut-out shapes onto a piece of paper lying on the floor in order to find creative compositions. Some artists have also attempted to induce randomness directly into their brains through the use of drugs.

Introducing randomness into your field of thought through the computer is infinitely safer than drugs and, paradoxically, it is controllable. The peculiar contribution the computer brings to randomness is that it allows the artist to select which aspects of his work are to remain under his conscious control and which aspects are to be random. Total randomness is available but is often unnecessary and undesirable. While the randomized aspects of my graphic output force me into creative consideration of new visual and expressive directions for my work, the criteria I use, or the criteria any other artist uses, to decide which of an apparently infinite number of aesthetic possibilities will be usable, are still essentially a mystery. The computer has not diminished and, I believe, will not diminish the importance of intuitive thinking in the creation of art. The computer can, however, set the stage for generative impulses within the artist.

After carefully considering my initial output, I either generate additional versions of the original designs, less randomness—more conscious control, or proceed directly to further manipulation through screenprinting. While my aesthetic goals are clearer at this point, I still do not have the completed work in mind. I approach screenprinting with a creative attitude by reconsidering my intuitions after each printed color. There may be as many as twenty colors in a single serigraph. By using transparent inks, I can control subtle variations in color by printing tinted glazes, one on top of another. This procedure allows me to "sneak up on" the effect I'm after and to achieve color richness and luminosity absent in opaque inks and single layers of transparent inks, too.

The coarse, mechanical, pointillistic pattern inherent in computer-controlled line printer graphics seems to unify the composition, enliven the color, and add a visual pulse to the work—all simultaneously. This is especially true if the computer pattern is eventful and printed as a glaze over previously printed patterns. The transparent glaze unifies the composition by

Route 740



subtly altering every other color in the work toward the glaze color. The pattern unifies by blanketing the work with a continuous texture. However, the pattern, itself being eventful, also enlivens the work with visual activity.

It may be stretching a point for me to consider myself a computer artist in a strict sense even though the computer plays a significant role in my work. While I use computer graphics as seed input for my screenprints, other artists attempt to solve *all* aesthetic problems and perform *all* graphic manipulation through programming. The completed art work is untouched after the plotter, printer, or CRT has completed its job. Craft has been completely usurped by numerically-controlled peripheral equipment. A physician friend of mine once derisively referred to this approach as "paraplegic" art. However heretical the idea, some handsome works have been produced. Jeff and Colette Bangert's plotter drawings are a case in point.

The implications of computer art may require us to change some of our basic ideas concerning the nature of art itself. Certainly the role of craft has been put in a strange light. The very fact that one can now create an art object without handwork reveals the making of art to be primarily a mental activity, not a craft-oriented activity, as some have supposed.

The idea of "original" drawing is also bent out of shape. A plotter drawing can be executed over and over again. The first drawing is only more "original" than the second in that it was executed first in time. In terms of aesthetic and graphic characteristics, they are both the same. Multiple original plotter drawings also differ significantly from limited edition prints in that with the computer we have multiple original art objects which are not printed impressions. Each drawing is a drawing.

A major contribution of the computer to art is its ability to inject randomness and surprise into the work.

When my work is viewed in this light, one can see ample opportunity for disagreement among those involved with computers and art. Fortunately, however, most of us see a multiplicity of approaches as healthy and desirable. We have not degenerated into an academic system which defines and dictates correct and incorrect ideas concerning computer usage in the arts.

Creative ideas are never stagnent, and I'm confident my work will continue to evolve. I hope to collaborate with the Department of Electrical Engineering at the University of Missouri in the development of software for teaching basic two- and three-dimensional design via interactive CRT display. The engineers have interactive hardware capable of displaying over four thousand discrete colors plus several black and white storage and refresh tubes. Where this will carry my art remains to be seen. But as the medium expands, new horizons cannot fail to open. ▼

A BOX OF BOOKS FOR ARTFUL COMPUTERS

Jonathan Benthall, *Science and Technology in Art Today* (New York: Praeger Publishers, 1972).

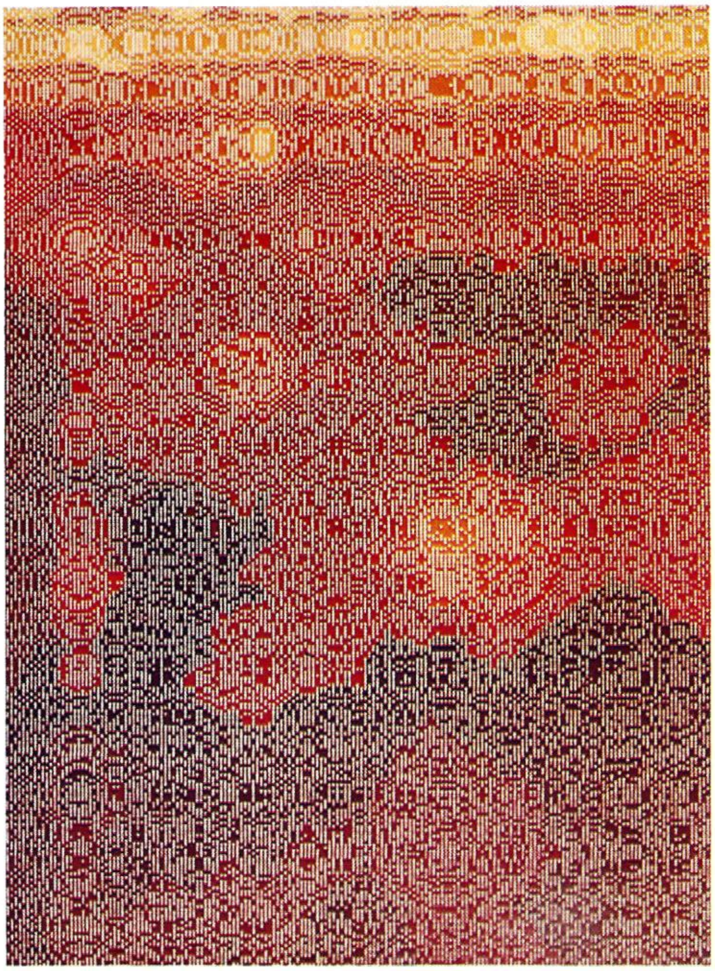
Douglas Davis, *Art in the Future* (New York: Praeger Publishers, 1973).

H.W. Franke, *Computer Graphics/Computer Art* (New York: Phaidon Publishers, Inc., 1971).

Jasia Reichardt, *The Computer in Art* (New York: Van Nostrand Reinhold Co., 1971).

C.H. Waddington, *Behind Appearances* (Cambridge, Massachusetts: The MIT Press, 1970).

Sun Dance II



Illustrated by Rex Ruden

Scott Joplin on Your Sci-Fi Hi-Fi

by Dorothy Siegel

I am a musically-inclined person with the kind of liberal arts background that didn't exactly saturate my education with an understanding of electronics and computers. As a clarinetist interested in contemporary music, I had performed musical compositions using electronically-produced sounds. But not only didn't I know a thing about how these sounds were produced, I was roundly intimidated by the strange symbols and unfamiliar language of the electronic computer world. Yet I knew I had to have at least a general appreciation of the subject if I was to fill the gap in my education. So I swallowed my fears, took curiosity by the hand, and signed up for a course in computer programming to teach me some basic concepts.

Last year, I plunged into programming books, 8080 instruction sets, and small computer magazines, in the hope that I'd absorb some of it, perhaps by osmosis. At roughly the same time, my husband (an electrical engineer) bought a microcomputer kit and started putting together our computer system.

Now one of the computer applications we both enjoy exploring is music synthesis. You, too, can get a computer to play music on an elementary level; at least it's much easier than one might think.

First let me suggest a simple coding scheme for musical tunes. It's a means for instructing the computer to convert the encoded music into "playable" form, and an interface unit to actually sound the music "played" by the computer. (The interface involves putting together a few electronic parts and a small speaker, or your hi-fi.)

Anyone can get a computer to play music.

Music can be defined as a pleasing arrangement of sounds. Each sound has certain principal attributes—pitch, duration, amplitude, and timbre or tone quality. For the purposes of this discussion:

- Pitch can be considered to be synonomous with frequency, and amplitude with loudness.
- The frequency of a sound is the number of times each second (cycles per second or Hertz) a periodic wave form repeats itself.
- The duration of a sound is the length of time it is sounded.
- The envelope of a sound is the variation in loudness over its duration.
- Timbre is a musician's way of describing the combination of different frequencies which together form one complex musical sound.

Control of a sound's pitch and duration are necessary if we are to define it at all. The ability to control envelope and timbre, although it increases musical interest, is not as fundamental. So for the first time around, let's leave out the complications.

The simplest wave form that a computer can produce is a square wave. This is because a square wave is a symmetrical alternation between only two values—e.g., 0 and 1. We will therefore use only a square wave as the periodic wave form whose frequency is the pitch of the musical sounds we produce. We further limit the range of playable frequencies to those from 262 Hz (middle C) to 1568 Hz (G#, almost three octaves higher). (By comparison, classical musical instruments produce sounds with frequencies ranging from approximately 25—4500 Hz, and the human ear can hear over a range of about 20—15,000 Hz.)

In general, the less one tries to specify, the easier it is to encode the music, program it, and run it. So, since we are controlling only the frequency and duration attributes of square waves, we can use a very simple coding scheme. In it, musical notes are specified by a four- or five-character string. Pitch is specified by the first three characters of the string:

First character: "A","B","C", "D","E","F","G". Indicates pitch within an octave.

Second character: "#" (sharp), "l" (flat)," " (natural). Indicates whether the pitch is raised, lowered, or neither, by a semitone.

At first I was roundly intimidated by the strange symbols and unfamiliar language of the electronic computer world.

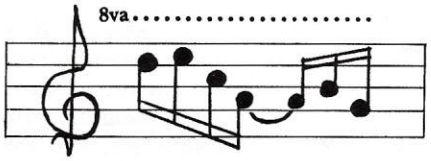
Third character: "1","2","3". Indicates the octave, where "1" is the lowest octave starting at A = 220Hz (Actually, the lowest allowable note is C = 262Hz.), "2" is the second octave starting at A = 440Hz, and "3" is the third octave starting at A = 880Hz.

Duration is specified by the fourth and fifth characters of the string:

Fourth character: "S","E","Q", "H","W". Indicates sixteenth note, eighth note, quarter note, half note, and whole note relative duration values.

Fifth character (optional): "." Indicates increase by half the duration of the note specified by the fourth character.

Thus, for example, the first six notes of Scott Joplin's "The Entertainer":



are coded as:

"D 3S","E 3S","C 3S","A 3E",
"B 3S","G 2E"

Once you have encoded the chosen musical tune properly, it is necessary to communicate this coded information to the computer in a form both you and the computer can understand. One of the easiest forms of human/machine communication is the high-level language BASIC. An advantage of using this language is that the many available versions of BASIC are similar to each other, so that a program written in one version can be run in another with only minor modifications.

The BASIC language program SCORE (printed below) contains both coded musical information and instructions for processing it. The musical information, coded as strings in the DATA statements, may use the convenient correspondence of one DATA statement to one musical measure.

A computer, when told to RUN the SCORE program, calculates a table of values which it places at a specified location in its memory. A brief assembly language routine, PLAY, (also printed below) is called, which uses these values to "play" the encoded music, outputting it to the interface. The speaker on the interface then outputs sound waves.

Let's pause a bit to think about how a hi-fi system produces sounds from a phonograph record. The phonograph cartridge converts the mechanical movement of the

COMPOSING FOR THE CURIOUS

In the tempered chromatic scale any pitch is the twelfth root of two ($\sqrt[12]{2}$) times the note a semitone below it. Using C = 220Hz as the reference frequency, we calculate the frequency (Fn) of a note *n* semitones above the reference frequency with Equation #1 below. An analysis of PLAY shows that the time of a half square wave cycle as a function of the pitch parameter (P) is given by Equation #2. To find the pitch parameter (P) we solve Equations #1 and #2 for P as given in Equation #3. The duration of a musical note is T1 multiplied by D. The duration parameter (D) is given by Equation #4.

P and D are rounded to the nearest integer and transferred to the score area (table of values) used by PLAY.

SCORE was written in Northstar BASIC.

PLAY was written in 8080 Assembly Language for a processor with a 2 MHz clock and no wait states.

Equations

Equation #1: $Fn = 220 \times (\sqrt[12]{2})^{n}$
$n = \#$ of semitones above "middle C" = 262 Hz

Equation #2: T1 = Time of half square wave cycle (in seconds) = $\frac{1}{2 \times Fn} = [(49 + 15 \times P)\ \text{states}] \times [(0.5 \times 10^{-6})\ \text{states/second}]$

Equation #3: $P = \frac{10^{-6}}{Fn \times 15} - - \frac{49}{15}$

Equation #4: $D = \frac{Fn}{T \times K6}$

Legend

Fn = Frequency of note
1/T = Duration of note, expressed as a fraction of a whole note
K6 = A constant used to adjust tempo

stylus, as it moves in the record grooves, to electrical impulses. The electrical impulses are amplified by the system's audio amplifier and then output to its speaker, which converts them into mechanical in-out movements of the speaker cone. These movements create waves in the air that our ears interpret as sound.

A sound output interface performs a function for the computer analogous to that performed by a hi-fi for phonograph records: it converts digital signals from the computer's output port to electrical impulses which its speaker in turn converts into mechanical movements, which then create sound waves.

The table of values calculated by SCORE, and used by PLAY to send appropriate digital impulses to the interface unit, consists of a one-byte pitch parameter (P) and a two-byte duration parameter (D) for each musical note. SCORE examines each string (musical note) in the DATA statements. For each string it calculates from the first three characters the number (*n*) of semitones the note is above the reference frequency; from the last one (or two) character(s) it calculates a number (T) that corresponds to the relative duration of each note.

I plunged into programming books and 8080 instruction sets in the hope that I'd absorb some concepts by osmosis.

An illegal character in a DATA statement will produce an error message and terminate score compilation. The error message will be in the form:

ERROR IN NOTE #1
DATA STRING "A #3P."
CHARACTER #4

In this example the fourth character is illegal. When the score is compiled without error messages, "SCORE COMPILATION COMPLETE!" will be indicated.

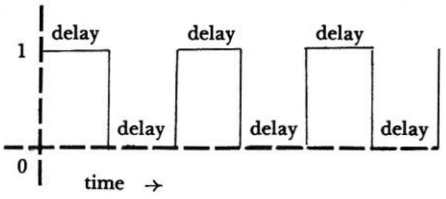
The starting location of the score area in SCORE is specified by the variable U. This location is given as 256 (100H), but it can be changed to any free memory location, provided the score location pointer in PLAY is made to agree.

A special end-of-score character ("X") is inserted as the last DATA statement in SCORE. When all DATA strings have been examined, and all P and D values transferred to the score area, SCORE reads the "X" and enters a zero pitch constant in the table, which is a signal to PLAY that it has reached the last value in the table.

A tune must be coded so it can be communicated to the computer in a form both you and the computer can understand.

After SCORE finishes calculating all values, it calls PLAY. This assumes that an assembled version of PLAY has already been loaded at location 0.

PLAY looks at the first P and D in the table. The value of P determines how many times a delay loop (LOOP1) executes. PLAY outputs a 1 to the output port (given as 24H), executes LOOP1, delays according to the value of P, then outputs a 0, delays, outputs a 1, delays, outputs a 0, and so on.



The frequency of the resulting square wave is that of the note we coded. The length of time this square wave is output corresponds to the note's duration. Each time a transition from 1 to 0 or 0 to 1 occurs, the note duration counter (set to D) is decremented by 1. When the counter is 0, the outputting of this square wave is finished, and the next P and D (for the next note) are taken from the table. This is repeated until PLAY encounters a pitch constant of "0", which indicates the end of the musical score.

The tempo of the music can be speeded up or slowed down by increasing or decreasing, respectively, the constant K6 in SCORE.

SCORE contains DATA statements for *The Entertainer* by Scott Joplin. Other music can be coded as DATA statements and inserted in SCORE instead of those used for the Joplin piece.

This is just one way to use fairly simple programs and a minimum of electronic hardware to generate musical tunes on your computer. The possibilities for musical expression are greatly enhanced when capabilities for different waveform generation and envelope and timbre control are added. Of course, in those cases, the hardware and programming requirements become correspondingly more elaborate. Still, once you've gone this far, you'll no doubt want to go on—if not immediately, then at least by the time you've played *The Entertainer* for the 541st time. ▼

FOR THOSE WHO WANT A MUSIC BOARD

Newtech Computer Systems, Inc. offers an S-100 bus compatible music board called the Model 6 Music Board. It features selectable output port address decoding, a latched six-bit digital-to-analog converter, audio amplifier, speaker, volume control, and an RCA phono jack for convenient connection to a home audio system. PLAY and SCORE are simplifications of programs presented in the Model 6 Music Board Users Manual. The Model 6 programs incorporate amplitude control for envelope shaping. The Music Board is capable of generating melodies, rhythms, sound effects, Morse code, touch-tone synthesis, and more.

The model 6 Music Board is availabe for $59.95 from local computer stores, or by mail order from Computer Mart of New York, Inc., 118 Madison Avenue, New York, New York 10016.

SCORE

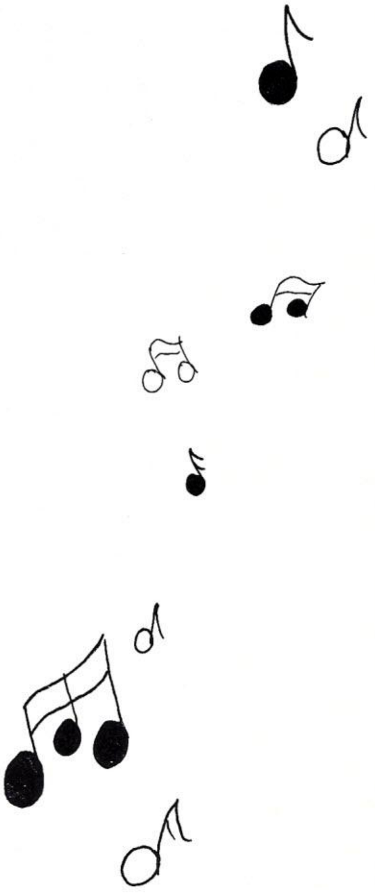
```
100 REM   TITLE \ SCORE
110 REM      THIS PROGRAM CALCULATES PITCH AND
115 REM      DURATION CONSTANTS FOR THE 8080
120 REM      ASSEMBLY LANGUAGE ROUTINE “PLAY”.
130 REM
140 REM
150 LET U=256 \ REM U DEFINES SCORE AREA IN MEMORY.
160 LET K1=2↑(1/12)
170 LET K6=.15 \REM TEMPO CONTROL
180   DIM Z$(5)
190 FOR V=1 TO 1000
200   LET C=1
210   READ Z$
220   LET N=100
230      IF Z$(1,1) = “A” THEN N=1
240      IF Z$(1,1) = “B” THEN N=3
250      IF Z$(1,1) = “C” THEN N=4
260      IF Z$(1,1) = “D” THEN N=6
270      IF Z$(1,1) = “E” THEN N=8
280      IF Z$(1,1) = “F” THEN N=9
290      IF Z$(1,1) = “G” THEN N=11
300      IF Z$(1,1) = “X” THEN GOTO 720
310       IF N=100 THEN GOTO 760
320   LET C=2
330   LET M=100
340      IF Z$(2,2) = “!” THEN M=N—1
350      IF Z$(2,2) = “#” THEN M=N+1
360      IF Z$(2,2) = “ “ THEN M=N
370       IF M=100 THEN GOTO 760
380   LET C=3
390   LET P=100
400      IF Z$(3,3) = “1” THEN P=M
410      IF Z$(3,3) = “2” THEN P=M+12
420      IF Z$(3,3) = “3” THEN P=M+24
430       IF P=100 THEN GOTO 760
440   LET C=4
450   LET T=100
460      IF Z$(4,4) = “S” THEN T=16
470      IF Z$(4,4) = “E” THEN T=8
480      IF Z$(4,4) = “Q” THEN T=4
490      IF Z$(4,4) = “H” THEN T=2
500      IF Z$(4,4) = “W” THEN T=1
510      IF T=100 THEN GOTO 760
520       IF LEN(Z$) = 4 THEN GOTO 560
530      LET C=5
540         IF Z$(5,5) = “.” THEN T=2*T/3
550   REM CALCULATE CONSTANTS
560      LET F1=220*(K1↑(P—1))
570      LET T1=10↑6/(2*F1)
580      LET K3=(T1—24.5)/7.5
```

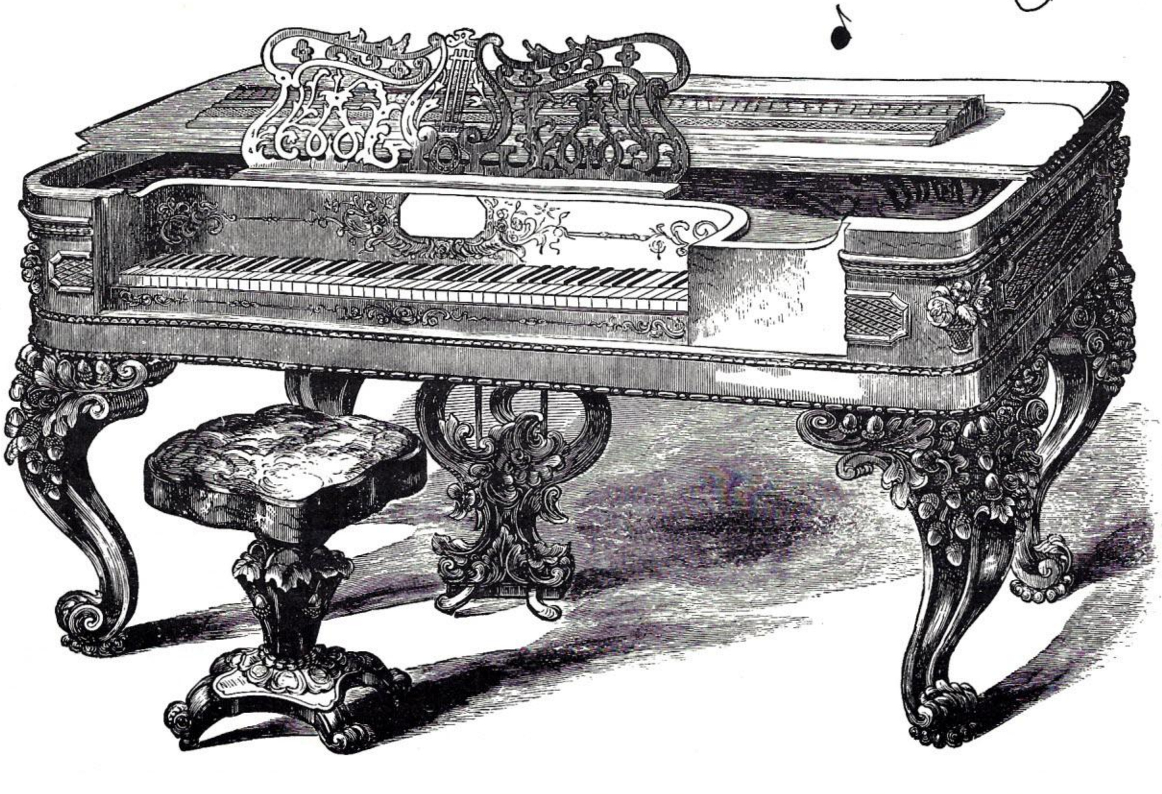
```
590         LET K4=F1/(K6*T)
600 LET D4=INT(K4) \ REM MAKE DURATION EVEN#
620 LET D5=INT(D4/256) \ REM CALC. 2 BYTES
630 LET D6=D5+1 \ REM D6=MSB
640 LET D7=D4+1—D5*256  \ REM D7=LSB
645 REM THE +1'S ARE ADJUSTMENTS FOR “PLAY”
646 REM ROUTINE COUNTER.
650 REM TRANSFER CONSTANTS TO SCORE AREA.
660 FILL U+3*(V—1),INT(K3+.5)
670 FILL U+3*(V—1)+1,D7
680 FILL U+3*(V—1)+2,D6
690 PRINT V,
700 NEXT V
710 STOP
720 FILL U+3*(V—1),0
730 PRINT
740 PRINT “SCORE COMPILATION COMPLETE!”
745 B1=CALL(3) \ REM USE BASIC STACK
750 STOP
760 PRINT “ERROR IN NOTE #”,V
770 PRINT “DATA STRING ”,Z$
780 PRINT “CHARACTER #”,C
790 STOP
800 END
810 REM
820 REM “THE ENTERTAINER” BY SCOTT JOPLIN
830   DATA “D 3S”.“E 3S”,“C 3S”,“A 3E”,“B 3S”,“G 2E”
840   DATA “D 2S”,“E 2S”,“C 2S”,“A 2E”,“B 2S”,“A 2S”,“A!2S”
850   DATA “G 1Q”,“G 3E”,“D 1S”,“d#1S”
860   DATA “E 1S”,“C 2E”,“E 1S”,“C 2E”,“E 1S”,“C 2Q.”
870   DATA “C 3S”,“D 3S”,“D#3S”
880   DATA “E 3S”,“C 3S”,“D 3S”,“E 3E”
890   DATA “B 3S”,“D 3E”
900   DATA “C 3Q.”,“D 1S”,“D#1S”
910   DATA “E 1S”,“C 2E”,“E 1S”,“C 2E”,“E 1S”,“C 2Q.”
920   DATA “C 2S”,“A 3S”,“G 2S”
930   DATA “F#2S”,“A 3S”,“C 3S”,“E 3E”,“D 3S”,“C 3S”,“A 3S”
940   DATA “D 3Q.”,“D 1S”,“D#1S”
950   DATA “E 1S”,“C 2E”,“E 1S”,“C 2E”,“E 1S”,“C 2Q.”
960   DATA “C 3S”,“D 3S”,“D#3S”
970   DATA “E 3S”,“C 3S”,“D 3S”,“E 3E”,“B 3S”,“D 3E”
980   DATA “C 3Q.”,“C 3S”,“D 3S”
990   DATA “E 3S”,“C 3S”,“D 3S”,“E 3E”,“C 3S”,“D 3S”,“C 3S”
1000   DATA “E 3S”,“C 3S”,“D 3S”,“E 3E”,“C 3S”,“D 3S”,“C 3S”
1010   DATA “E 3S”,“C 3S”,“D 3S”,“E 3E”,“B 3S”,“D 3E”
1020   DATA “C 3Q”,“C 3S”,“E 2S”,“F 2S”,“F#2S”
1030   DATA “G 2E”,“A 3S”,“G 2E”,“E 2S”,“F 2S”,“F#2S”
1040   DATA “G 2E”,“A 3S”,“G 2E”,“E 2S”,“C 2S”,“G 1S”
1050   DATA “A 2S”,“B 2S”,“c 2S”,“D 2S”,“E 2S”,“D 2S”,“C 2S”,“D 2S”
1060   DATA “C 2E”,“G 1E”,“C 1E”,“X”
READY
```

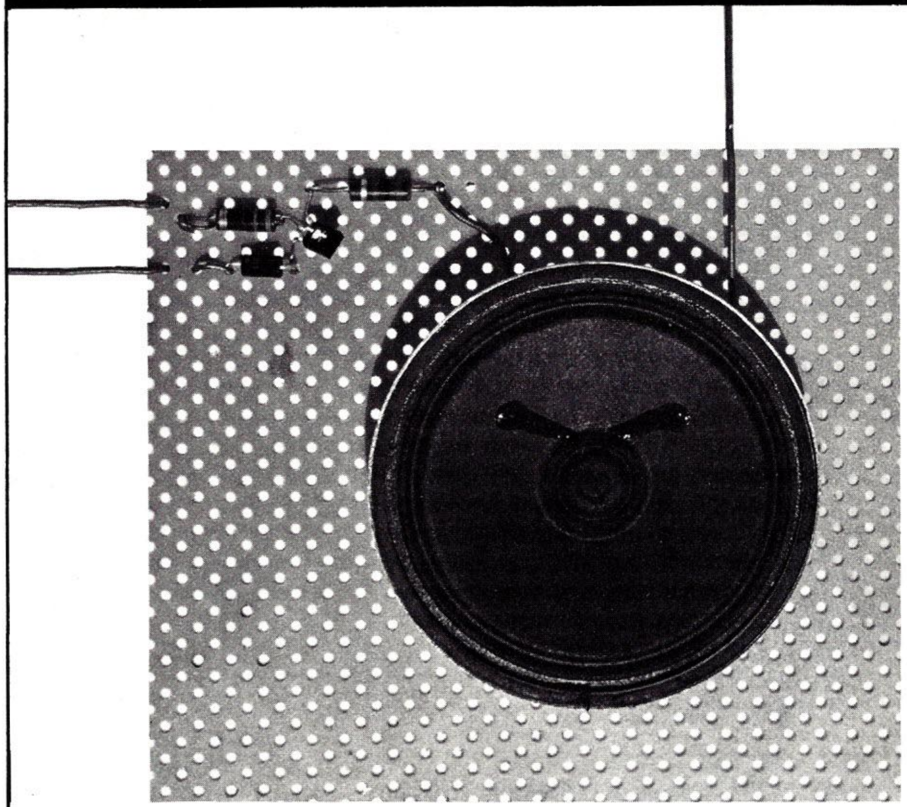
PLAY

```
0000                ;TITLE: PLAY
0000                ; DESCRIPTION: PLAY STARTS AT THE
0000                ; BEGINNING OF THE MEMORY AREA
0000                ; DESIGNATED SCORE, AND TRANSFERS
0000                ; INTO THE PLAY ROUTINE A 1-BYTE
0000                ; PITCH PARAMETER AND A 2-BYTE DURATION
0000                ; PARAMETER. THE ROUTINE THEN OUTPUTS TO
0000                ; THE DESIGNATED PORT THE MUSICAL NOTE
0000                ; SPECIFIED BY THESE NOTE PARAMETERS.
0000                ; PLAY CONTINUES TRANSFERRING NOTE
0000                ; PARAMETERS AND OUTPUTTING NOTES
```

```
0000                 ;  UNTIL A PITCH CONSTANT OF ZERO
0000                 ;  IS ENCOUNTERED WHICH INDICATES
0000                 ;  THE END OF THE MUSICAL SCORE.
0000                 ;     THIS VERSION OF PLAY WAS
0000                 ;  WRITTEN FOR A 2MHZ 8080 HAVING
0000                 ;  NO WAIT STATES BUT CAN BE MODIFIED
0000                 ;  FOR WAIT STATES OR FOR Z80'S.
0000  31 3A 00       PLAY    LXI   SP,STACK  ;INITIALIZE STACK POINTER
0003  21 00 01       INIT    LXI   H,SCORE   ;INITIALIZE SCORE POINTER
0006  97             NEXT    SUB   A         ;ZERO ACCUMULATOR
0007  BE                     CMP   M         ;TEST FOR PITCH PARAMETER =
0008  C8                     RZ              ;ZERO. RETURN IF TRUE.
0009  4E                     MOV   C,M       ;TRANSFER NEXT PITCH
000A                 ;                       ;PARAMETER TO C REGISTER.
000A  23                     INX   H
000B  5E                     MOV   E,M       ;TRANSFER DURATION PARAMETER
000C  23                     INX   H         ;TO DE REGISTER PAIR.
000D  56                     MOV   D,M
000E  23                     INX   H
000F  CD 15 00               CALL  TONE      ;CALL TONE PLAYING ROUTINE.
0012  C3 06 00               JMP   NEXT
0015                 ;
0015  2F             TONE    CMA             ;TOGGLE OUTPUT BIT.
0016  41                     MOV   B,C       ;COPY PITCH CONSTANT TO B.
0017  D3 24                  OUT   PORT      ;OUTPUT TO DESIRED PORT.
0019  05             LOOP1   DCR   B         ;DELAY ACCORDING TO
001A  C2 19 00               JNZ   LOOP1     ;PITCH PARAMETER IN B.
001D  1D                     DCR   E         ;DECREMENT DURATION
001E  C2 26 00               JNZ   LOOP2     ;COUNTER IN D,E.
0021  15                     DCR   D
0022  C2 15 00               JNZ   TONE      ;DO NEXT HALF-CYCLE
0025  C9                     RET             ;RET WHEN DURATION COUNT=0.
0026  7F             LOOP2   MOV   A,A       ;WASTE TIME
0027  C3 15 00               JMP   TONE
002A                 ;
002A                 STACK   EQU   $+10H
002A                 SCORE   EQU   100H      ;YOUR SCORE AREA?
002A                 PORT    EQU   24H       ;YOUR OUTPUT PORT?
002A                 SP      EQU   6
%
```







BUILDING A BASIC MUSIC BOARD

by Eben F. Ostby based on the design of Dorothy Siegel

If you want to build your own output device for producing music on your computer, you'll find it a lot easier than assembling an IMSAI, even if you can't tell one component from the next. All the parts are available at your local Radio Shack or similar electronics shop, and the total cost shouldn't be more than a few dollars. To show you that anyone can do it, *ROM* did it in about half an hour last night, and we're not engineers, either.

First, take a piece of breadboard. It's thin plastic sheet with evenly spaced holes drilled in it to hold the component leads or wires. Or you can get some of the new solderless breadboard, which allows you to put the circuit together by simply pushing the wires into adjacent holes. By reversing the process (pulling) you can then readily disassemble the project. This type of breadboard is usually used for temporary circuits, but if you've never soldered before, you might find them convenient. If you're using the standard breadboard, you'll need a low-heat soldering iron and solder.

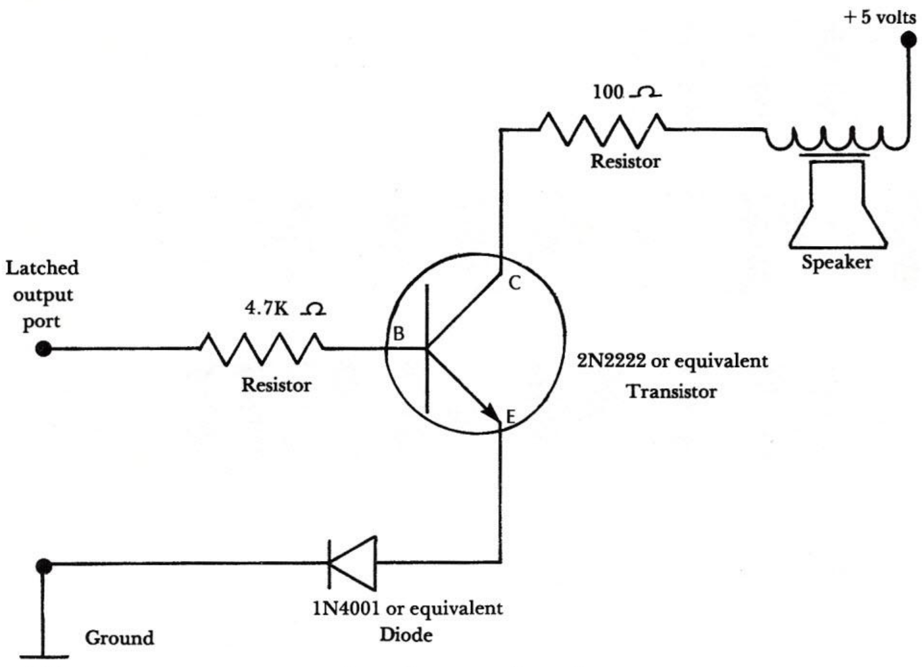
Cut a piece of hook-up wire a foot or so long. Strip about one-half inch of insulation off each end. Then weave one end through two holes on the board so the wire won't pull out easily. Align the wire so one stripped end just pokes through one of the holes, about a quarter inch. (The other end is for later.)

Take the 4.7K ohm resistor (it'll probably have yellow, violet, and red stripes on it, in that order) and poke one wire down the same hole as the hook-up wire you previously stripped. Twist the two wires together at the back of the board and solder. Clip off any extra wire that pokes through too far. Take the other end of the resistor, poke it through a nearby hole. Find the base lead of the transistor, push it through the same hole—and twist.

The package that the transistor came in may say which lead is the base lead or the transistor itself may be lableled with the letters E (for emitter), B (for base), and C (for collector). You can also find guide cards with drawings of transistors in many electronics stores. Other sources of pictorial information include backs of those $1.59 "bargain packs" of miscellaneous components, and electronics hobby books.

When you've found the base, solder it to the free end of the resistor the way you soldered the wire to the other end. Making sure the wires don't touch where they're not supposed to, attach the hundred-ohm resistor (brown, black, brown) to the collector lead of the transistor, solder it. Next, attach a short (three-inch or so) piece of wire to the other end of the resistor, and solder. The diode, which is attached next, must face the right way. Attach the end of the diode—where the tail of the arrow is, or where the little band *isn't*—to the emitter terminal of the transistor. The other end of the diode—the one where the arrow points, or where the band is, is soldered to another piece of wire about a foot long. Again, make sure you thread this wire through a couple of holes on the board if you can; that way it won't pull out. By the way, the arrangement of these parts on the board isn't really critical, but it's a good idea to arrange them like the schematic drawing of the circuit.

Now attach the little speaker to the board. If you bought one you can attach directly, there's no problem. Ours didn't so we cut a hole in the board just big enough to press the

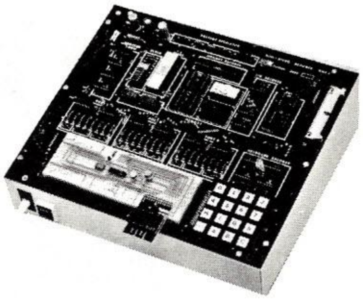


Sound output interface

magnet or back part of the speaker through, and small enough so that it fitted snugly and stayed there. If you have a big speaker, you may want to attach the board to the speaker, instead of vice-versa. Now take the end of the short piece of wire, the one that you soldered to the hundred-ohm resistor, and solder it to one of the speaker's terminals. Take another piece of wire about a foot long, and solder it to the other speaker terminal (if there aren't exactly two terminals on your speaker, find out why not. Somewhere are the two correct ones.) The other end of this wire should be stripped, as well.

Now you've finished it. To attach it to your computer, take the first wire you soldered—the one that goes to the 4.7K ohm resistor—and attach it to an output port on your computer. This can be one on an existing peripheral, or it can be one you use just for the music interface. Don't solder this, though, since some day you'll want to remove it. The second wire, the one from the diode, goes to a ground point on the computer. The last one, from the speaker, goes to a five volt source, which is the standard voltage running around in your computer. Look in the computer's manual to find where all these points are, and then attach the wires. Now you're hooked up. If you have an "unlatched" output port instead of a "latched" one, you can try that, too—it should sound different, but work the same way. A latched signal turns on, and stays on until it's turned off. An unlatched signal turns on, but doesn't stay on—it slowly fades out. This means that the shape of the wave produced is different, though tonally it's the same.

When you try to get the programs to run, remember that your software may be different from this. The BASIC program is written in North Star BASIC, which has a FILL instruction letting you put data directly into memory—you tell it where. The CALL function lets you call an assembly-language procedure directly. CALL (3), in this case, starts executing the assembly-language procedure at location 3. You may have to find an equivalent way of doing these things if your system is different. But it shouldn't be difficult and soon your computer will be playing away to its heart's content. ▼



The $422.50* Microprocessor. Complete.

Are we bugs?

The MMD-1 is your ticket to the world of microprocessors. It's a complete microcomputer system. And just as important, it comes with the industry's most advanced instructional software – 700 pages by Rony/Larsen/Titus, authors of the famous BUGBOOK® series.

Without any prior knowledge of electronics you can be up and operating in a matter of hours. Teaching yourself everything from fundamental logic to sophisticated interfacing.

And you'll be learning on the most complete hardware package of its kind. Direct Keyboard Entry of Data . . . Built-in Power Supply . . . Direct Access to Output Ports . . . Monitoring of Address and Data Busses . . . Unique Breadboarding Facilities for Interfaces . . . and . . . more.

The MMD-1 is clearly the best buy in the industry. And it's available now at your nearest computer store. Stop in. Or write us for the store nearest you and a full 8 page illustrated brochure.

*Suggested resale price.

E&L INSTRUMENTS, INC.
61 First Street, Derby, Conn. 06418
(203) 735-8774 Telex No. 96 3536
Dept. R

THE COMPULSIVE PROGRAMMER

by Joseph Weizenbaum

There is a distinction between physically embodied machines, whose ultimate function is to transduce energy or deliver power, and abstract machines, i.e., machines that exist only as ideas. The laws which the former embody must be a subset of the laws that govern the real world. The laws that govern the behavior of abstract machines are not necessarily so constrained. One may, for example, design an abstract machine whose internal signals are propagated among its components at speeds greater than the speed of light, in clear violation of physical law. The fact that such a machine cannot actually be built does not prohibit the exploration of its behavior. It can be thought about and even simulated on a computer. (Indeed, the Education Research Center at M.I.T. has made computer-generated films that enable viewers to observe a world in which vehicles travel at physically impossible speeds.) The human imagination must be capable of transcending the limitations of physical law if it is to be able to conceive such law at all.

The computer is, of course, a physically embodied machine and, as such, cannot violate natural law. But it is not completely characterized by only its manifest interaction with the real world. Electrons flow through it, its tapes move, and its lights blink, all in strict obedience to physical law, to be sure, and the courses of its internal rivers of electrons are determined by openings and closings of gates, that is, by physical events. But the game the computer plays out is regulated by systems of ideas whose range is bounded only by the limitations of the human imagination. The physically determined bounds on the electronic and mechanical events internal to the computer do not matter for that game—any more than it matters how tightly a chess player grips his bishop or how rapidly he moves it over the board.

A computer running under control of a stored program is thus detached from the real world in the same way that every abstract game is. The chess board, the thirty-two chessmen, and the rules of chess constitute a world entirely separate from every other world. So does a computer system together with its operating manual. The chess player who has made a bad move cannot explain his error away by pointing to some external empirical fact that he could not have known but that, had he known it, would have led him to make a better move. Neither may a programmer whose program behaves differently from what he had intended look for the fault anywhere but in the game he has himself created. He may have misread the computer system's manuals or otherwise have misunderstood his compu-

From *Computer Power and Human Reason,* Copyright © 1976 by W. H. Freeman and Co.

Illustrated by Cindy Hain

C. HAIN

ting environment, just as a novice chess player may misread the rules for, say, castling. But no datum existing in the world outside the computer system he is using can be at all relevant to the behavior of the world he has created. A computer's failure to behave exactly as its programmer intends cannot even be attributable solely to some limitation unique to the computer. In effect, every general-purpose computer is a kind of universal machine that can in principle do what any other general-purpose computer can do. In this important sense, a specific general-purpose computer has no limitations unique to it. The computer, then, is a playing field on which one may play out any game one can imagine. One may create worlds in which there is no gravity, or in which two bodies attract each other, not by Newton's inverse-square law, but by an inverse-cube (or *nth*-power) law, or in which time dances forward and backward in obedience to a choreography as simple or complex as one wills. One can create societies in whose economies prices rise as goods become plentiful and fall as they become scarce, and in which homosexual unions alone produce offspring. In short, one can singlehandedly write and produce plays in a theater that admits of no limitations. And, what is most important, one need know only what can be inferred directly from one's computer-system manual or constructed by one's own imagination.

The computer programmer is a creator of universes for which he alone is the lawgiver.

By way of contrast, let us consider the act of designing a computer circuit. Last month I displayed a circuit diagram for a one-bit adder. One may build the corresponding adder for a machine whose basic cycle time is, say, one microsecond, in other words, for a machine that experiences a million quiescent-active time-interval pairs every second. Suppose such an adder is built and is found to function properly. Say the adder's designers later install it in a machine that operates at ten times the rate of the original machine, that, so to speak, turns itself on and off ten million times per second, and suppose that the adder doesn't work in its new environment. What can be the trouble? More importantly, what sources of knowledge may have to be tapped in order to correctly diagnose the disorder? Clearly the rules of the game as stated by the abstract equations governing the behavior of AND, OR, and NOT gates and by the adder's circuit diagram are not sufficient. Nothing has changed with respect to the adder's abstract design. By hypothesis, all that has changed is the rate at which the adder is being exercised. The new rate must therefore be responsible for the adder's malfunctioning. In order to discover how to repair the adder or, indeed, to know whether it is physically possible to operate any device at such a high rate at all, one must apply some relevant knowledge of the physical world.

This example, though simplified, is not fanciful, and it is worth pursuing a little further. Nearly everyone knows that violin strings (and other objects as well) have natural (or resonant) frequencies of vibration. The A string, for example, vibrates naturally approximately 435 times per second. If two A strings are near one another, and one of them is made to vibrate at its natural frequency, say, by plucking it, then the other will also be set into vibration (at, of course, the same frequency). If, however, the first string is forced to vibrate at some frequency other than its resonant one, then, even though both are A strings, the second will not vibrate. In effect, strings are both transmitting and receiving antennas at their natural acoustic frequencies. With this in mind, suppose two children living across the street from one another set up a signaling system consisting of two parallel strings bridging the street and connecting their two houses. To signal the arrival of father, say, one of the strings is made to vibrate, as is the other string to signal the arrival of mother. If, however, one string is made to vibrate at its natural frequency, then the other string will be induced to vibrate as well. The system will therefore fail to function as intended; it will be incapable of announcing the arrival of only one parent.

Electric circuits are somewhat like acoustic systems in this respect, although the frequencies associated with them are very much higher than those of their acoustic counterparts. In particular, circuits have natural (or resonant) frequencies. Indeed, many may be tuned to resonate at quite specifically chosen frequencies, in order to act, for example, as antennas for broadcasting stations. Home radio receivers have similarly tunable circuits that act as receiving antennas. The trouble with our adder may be that, although none of its circuits resonated so as to broadcast and receive signals from one another when they were operated at a frequency of a million pulses per second, they did act as broadcasting and receiving antennas at the higher operating frequency. They consequently confused the information they were designed to manipulate.

Properly trained electrical engineers, of course, command the theory from which such phenomena can be deduced. Others may identify the trouble on the basis of earlier experiences. But the engineer who has neither the theory nor the experience could never discover the cause of the trouble by deductive logic alone. He could repair his circuit only by fortuitous tinkering, or by receiving outside help, or by, in effect, repeating the creative scientific work of a man like Heinrich Hertz, the discoverer of the electromagnetic radiation phenomenon to which we are here alluding. But we label Hertz's work creative precisely because it involves observations of phenomena not mentioned in any existing manuals and because it infers general laws from the particulars observed. That is quite the opposite of deduction.

An engineer is inextricably impacted in the material world. His creativity is confined by its laws; he may, finally, do only what may lawfully be done. But he is doomed to exercise his trade in a Kafkaesque castle from which there is, even in principle, no escape. For he cannot know the whole plan that determines what rooms there are in the world, what doors exist between them, and how the doors may be opened. When a device an engineer has designed doesn't work, he therefore cannot always know, or tell by his own reasoning alone, whether he is in an antechamber to success where only his blunders keep him from opening its doors, or whether he has wandered into a closet from which there is no exit. Then he must appeal to others, his teachers, his colleagues, his books, to tell him, or at least to hint at, a formula that will compel the insouciant attendant (nature) to lead him out and on.

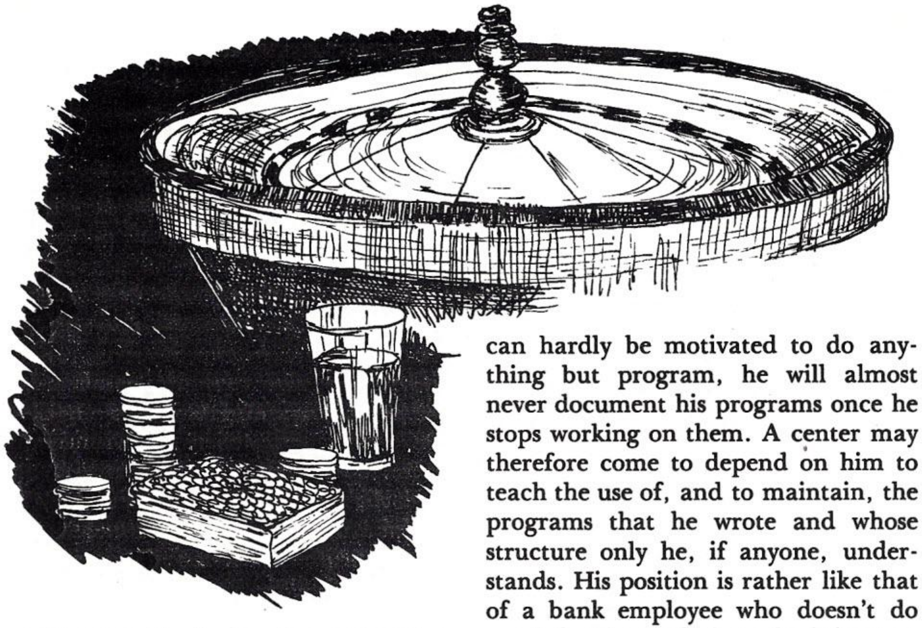
Programming is often abandoned precisely when it ceases to be purely incestuous.

The computer programmer, however, is a creator of universes for which he alone is the lawgiver. So, of course, is the designer of any game. But universes of virtually unlimited complexity can be created in the form of computer programs. Moreover, and this is a crucial point, systems so formulated and elaborated *act out* their programmed scripts. They compliantly obey their laws and vividly exhibit their obedient behavior. No playwright, no stage director, no emperor, however powerful, has ever exercised such absolute authority to arrange a stage or a field of battle and to command such unswervingly dutiful actors or troops.

One would have to be astonished if Lord Acton's observation that power corrupts were not to apply in an environment in which omnipotence is so easily achievable. It does apply. And the corruption evoked by the computer programmer's omnipotence manifests itself in a form that is instructive in a domain far larger than the immediate environment of the computer. To understand it, we will have to take a look at a mental disorder that, while actually very old, appears to have been transformed by the computer into a new genus: the compulsion to program.

Wherever computer centers have become established, that is to say, in countless places in the United States, as well as in virtually all other industrial regions of the world, bright young men of disheveled appearance, often with sunken glowing eyes, can be seen sitting at computer consoles, their arms tensed and waiting to fire their fingers, already poised to strike, at the buttons and keys on which their attention seems to be as riveted as a gambler's on the rolling dice. When not so transfixed, they often sit at tables strewn with computer printouts over which they pore like possessed students of a cabalistic text. They work until they nearly drop, twenty, thirty hours at a time. Their food, if they arrange it, is brought to them: coffee, Cokes, sandwiches. If possible, they sleep on cots near the computer. But only for a few hours—then back to the console or the printouts. Their rumpled clothes, their unwashed and unshaven faces, and their uncombed hair all testify that they are oblivious to their bodies and to the world in which they move. They exist, at least when so engaged, only through and for the computers. These are computer bums, compulsive programmers. They are an international phenomenon.

How may the compulsive programmer be distinguished from a merely dedicated, hard-working professional programmer? First, by the fact that the ordinary professional programmer addresses himself to the problem to be solved, whereas the compulsive programmer sees the problem mainly as an opportunity to interact with the computer. The ordinary computer programmer will usually discuss both his substantive and his technical programming problem with· others. He



will generally do lengthy preparatory work, such as writing and flow diagramming, before beginning work with the computer itself. His sessions with the computer may be comparatively short. He may even let others do the actual console work. He develops his program slowly and systematically. When something doesn't work, he may spend considerable time away from the computer, framing careful hypotheses to account for the malfunction and designing crucial experiments to test them. Again, he may leave the actual running of the computer to others. He is able, while waiting for results from the computer, to attend to other aspects of his work, such as documenting what he has already done. When he has finally composed the program he set out to produce, he is able to complete a sensible description of it and to turn his attention to other things. The professional regards programming as a means toward an end, not as an end in itself. His satisfaction comes from having solved a substantive problem, not from having bent a computer to his will.

The compulsive programmer is usually a superb technician, moreover, one who knows every detail of the computer he works on, its peripheral equipment, the computer's operating system, etc. He is often tolerated around computer centers because of his knowledge of the system and because he can write small subsystem programs quickly, that is, in one or two sessions of, say, twenty hours each. After a time, the center may in fact be using a number of his programs. But because the compulsive programmer can hardly be motivated to do anything but program, he will almost never document his programs once he stops working on them. A center may therefore come to depend on him to teach the use of, and to maintain, the programs that he wrote and whose structure only he, if anyone, understands. His position is rather like that of a bank employee who doesn't do much for the bank, but who is kept on because only he knows the combination to the safe. His main interest is, in any case, not in small programs, but in very large, very ambitious systems of programs. Usually the systems he undertakes to build, and on which he works feverishly for perhaps a month or two or three, have very grandiose but extremely imprecisely stated goals. Some examples of these ambitions are: new computer languages to facilitate man-machine communication; a general system that can be taught to play any board game; a system to make it easier for computer experts to write super-systems (this last is a favorite). It is characteristic of many such projects that the programmer can long continue in the conviction that they demand knowledge about nothing but computers, programming, etc. And that knowledge he, of course, commands in abundance. Indeed, the point at which such work is often abandoned is precisely when it ceases to be purely incestuous, i.e., when programming would have to be interrupted in order that knowledge from outside the computer world may be acquired.

Unlike the professional, the compulsive programmer cannot attend to other tasks, not even to tasks closely related to his program, during periods when he is not actually operating the computer. He can barely tolerate being away from the machine. But when he is nevertheless forced by circumstances to be separated from it, at least he has his computer printouts with him. He studies them, he talks about them to anyone who will listen—though, of course, no one else can understand them. Indeed, while in the grip of his compulsion, he can talk of nothing but his program. But the only time he is, so to say, happy is when he is at the computer console. Then he will not converse with anyone but the computer. We will soon see what they converse about.

The compulsive programmer spends all the time he can working on one of his big projects. "Working" is not the word he uses; he calls what he does "hacking." To hack is, according to the dictionary, "to cut irregularly, without skill or definite purpose; to mangle by or as if by repeated strokes of a cutting instrument." I have already said that the compulsive programmer, or hacker as he calls himself, is usually a superb technician. It seems therefore that he is not "without skill" as the definition would have it. But the definition fits in the deeper sense that the hacker is "without definite purpose": he cannot set before himself a clearly defined long-term goal and a plan for achieving it, for he has only technique, not knowledge. He has nothing he can analyze or synthesize; in short, he has nothing to form theories about. His skill is therefore aimless, even disembodied. It is simply not connected with anything other than the instrument on which it may be exercised. His skill is like that of a monastic copyist who, though illiterate, is a first-rate calligrapher. His grandiose projects must therefore necessarily have the quality of illusions, indeed, of illusions of grandeur. He will construct the one grand system in which all other experts will soon write their systems.

Grandiose projects must necessarily have the quality of illusions, indeed, of illusions of grandeur.

(It has to be said that not all hackers

are pathologically compulsive programmers. Indeed, were it not for the often, in its own terms, highly creative labor of people who proudly claim the title "hacker," few of today's sophisticated computer time-sharing systems, computer language translators, computer graphics systems, etc., would exist.)

Programming systems can, of course, be built without plan and without knowledge, let alone understanding, of the deep structural issues involved, just as houses, cities, systems of dams, and national economic policies can be similarly hacked together. As a system so constructed begins to get large, however, it also becomes increasingly unstable. When one of its subfunctions fails in an unanticipated way, it may be patched until the manifest trouble disappears. But since there is no general theory of the whole system, the system itself can be only a more or less chaotic aggregate of subsystems whose influence on one another's behavior is discoverable only piecemeal and by experiment. The hacker spends part of his time at the console piling new subsystems onto the structure he has already built—he calls them "new features"—and the rest of his time in attempts to account for the way in which substructures already in place misbehave. That is what he and the computer converse about.

The psychological situation the compulsive programmer finds himself in while so engaged is strongly determined by two apparently opposing facts: first, he knows that he can make the computer do anything he wants it to do; and second, the computer constantly displays undeniable evidence of his failures to him. It reproaches him. There is no escaping this bind. The engineer can resign himself to the truth that there are some things he doesn't know. But the programmer moves in a world entirely of his own making. The computer challenges his power, not his knowledge.

Indeed, the compulsive programmer's excitement rises to its highest, most feverish pitch when he is on the trail of a most recalcitrant error, when everything ought to work but the computer nevertheless reproaches him by misbehaving in a number of mysterious, apparently unrelated ways. It is then that the system the programmer has himself created gives every evidence of having taken on a life of its own and, certainly, of having slipped from his control. This too is the point at which the idea that the computer can be "made to do anything" becomes most relevant and most soundly based in reality. For, under such circumstances, the misbehaving artifact is, in fact, the programmer's own creation. Its very misbehavior can, as we have already said, be the consequence only of what the programmer himself has done. And what he has done he can presumably come to understand, to undo, and to redo to better serve his purpose. Accordingly his mood and his activity become frenzied when he believes he has finally discovered the source of the trouble. Should his time at the console be nearly up at that moment, he will take enormous risks with his program, making substantial changes, one after another, in minutes or even seconds without so much as recording what he is doing, always pleading for just another minute. He can, under such circumstances, rapidly and virtually irretrievably destroy weeks and weeks of his own work. Should he, however, find a deeply embedded error, one that actually does account for much of the program's misbehavior, his joy is unbounded. It *is* a thrill to see a hitherto moribund program suddenly come back to life; there is no other way to say it. When some deep error has been found and repaired, many different portions of the program, which until then had given nothing but incomprehensible outputs, suddenly behave smoothly and deliver precisely the intended results. There is reason for the diagnostician to be pleased and, if the error was really deep inside the system, even proud.

His apparently devoted efforts to improve and promote his own creation are really an assault on it.

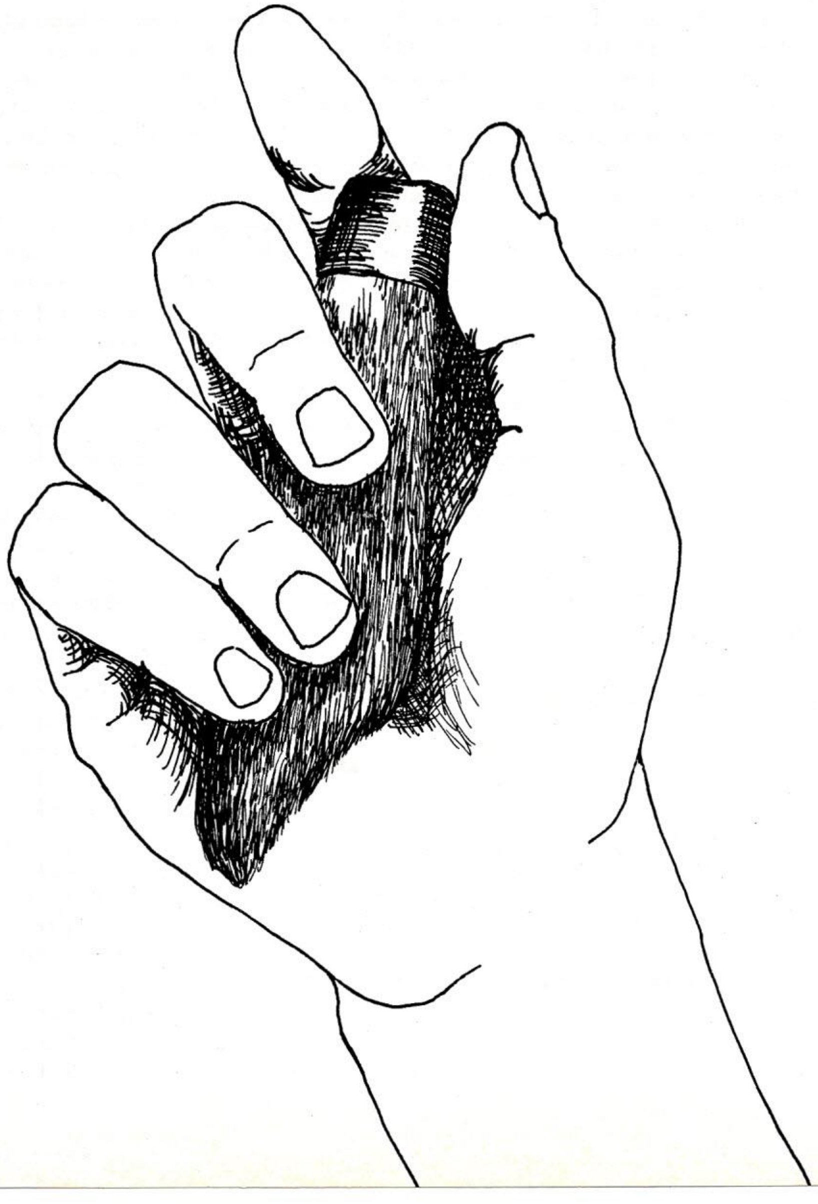
But the compulsive programmer's pride and elation are very brief. His success consists of his having shown the computer who its master is. And having demonstrated that he can make it do this much, he immediately sets out to make it do even more. Thus the entire cycle begins again. He begins to "improve" his system, say, by making it run faster, or by adding "new features" to it, or by improving the ease with which data can be entered into it and gotten out of it. The act of modifying the then-existing program invariably causes some of its substructures to collapse; they constitute, after all, an amorphous collection of processes whose interactions with one another are virtually fortuitous. His apparently devoted efforts to improve and promote his own creation are really an assault on it, an assault whose only consequence can be to renew his struggle with the computer. Should he be prevented from so sabotaging his own work, say, by administrative decision, he will become visibly depressed, begin to sulk, display no interest in anything around him, etc. Only a new opportunity to compute can restore his spirit.

It must be emphasized that the portrait I have drawn is instantly recognizable at computing installations all over the world. It represents a psychopathology that is far less ambiguous than, say, the milder forms of schizophrenia or paranoia. At the same time, it represents an extremely developed form of a disorder that afflicts much of our society.

How are we to understand this compulsion? We must first recognize that it *is* a compulsion. Normally, wishes for satisfaction lead to behaviors that have a texture of discrimination and spontaneity. The fulfillment of such wishes leads to pleasure. The compulsive programmer is driven; there is little spontaneity in how he behaves; and he finds no pleasure in the fulfillment of his nominal wishes. He seeks reassurance from the computer, not pleasure. The closest parallel we can find to this sort of psychopathology is in the relentless, pleasureless drive for reassurance that characterizes the life of the compulsive gambler.

The compulsive gambler is also to be sharply distinguished from the professional gambler. The latter is, in an important sense, not a gambler at all. (We may leave aside the cheater and the professional confidence man, for certainly neither of them are gamblers either.) The so-called professional gambler is really an applied statistician, and perhaps an applied psychologist as well. His income depends in almost no way on luck. He knows applied probability theory, and he uses it to calculate odds and then to play those odds in such combinations and aggregates that he can predict his income during a period of, say, a year with almost mathematical certainty. That is not gambling. Then there are people who gamble but who are neither professional nor compulsive gamblers. To the compulsive gambler, gambling, the game, is everything. Even winning is less important than playing. He is, so to say, happy only when he is at the gambling table.

The compulsive gambler believes himself to be in control of a magical world.



Anyone who has ever worked in a computer center or a gambling casino that closes its doors at night will recognize the scene described by Dostoevski, himself a passionate gambler, in *The Gambler:*

> "By eleven o'clock, there remain at the roulette table only those desperate players, the real gamblers, for whom there exists but the roulette table, . . . who know nothing of what is going on around them and take no interest in any matters outside the roulette saloon, but only play and play from morning till night, and would gladly play all round the clock if it were permitted. These people are always annoyed when midnight comes, and they must go home, because the roulette bank is closed. And when the chief croupier, about twelve o'clock, just before the close calls out, 'The last three turns, gentlemen!' these men are ready to stake all they have in their pockets on those last three turns, and it is certain that it is just then that these people lose most."

Dostoevski might as well have been describing a computer room.

The medical literature on compulsive gambling concerns itself mainly with the psychogenesis of that compulsion, and then almost entirely from a psychoanalytic perspective. But I need not recapitulate the psychoanalytic argument for my purposes here. It is enough to say here that psychoanalysts, beginning with Freud, saw megalomania and fantasies of omnipotence as principal ingredients in the psychic life of the compulsive gambler. We do not have to accept, or reject, psychoanalytic accounts of the origins of such fantasies—e.g., that they are rooted in unresolved Oedipal conflicts leading to wishes to overpower the father that in turn lead to unconscious motivations to fail—in order to join the psychoanalysts and novelists like Dostoevski in seeing the central role that megalomaniac fantasies of omnipotence play in compulsive gambling.

The gambler, according to the psychoanalyst Edmund Bergler, has three principal convictions: first, he is subjectively certain that he will win; second, he has an unbounded faith in his own cleverness; third, he knows that life itself is nothing but a gamble.

What grounds can there possibly be for knowing that one will win a game of pure chance? To know that the roll of a pair of dice or the turn of a card is a purely chance event is to know that nothing one does can possibly affect the outcome. There precisely is the rub! The compulsive gambler believes himself to be in control of a magical world to which only few men are given entrance. "He believes," writes Bergler, "Fate has singled him out . . . and communicates with him by means of small signs indicating approval and reproach." The gambler is the scientist of this magical world. He is the interpreter of the signs that Fate communicates to him, just as the scientist in the real world is an interpreter of the signs that nature communicates to everyone who cares to become sensitive to them. And like the natural scientist, the compulsive gambler always has a tentative hypothesis that accounts for almost all the signs he has so far observed, that, in other words, constitutes a very nearly complete picture of those aspects of the universe which interest him. The test of the adequacy of both the scientist's and the magician's view of the world is its power to predict and, under suitably arranged conditions, to control. Hence, according to Bergler, the compulsive gambler sees himself as "not the victim, but the executive arm of unpredictable chance."

So that's it: Touch a hunchback, carry a rabbit's foot, don't cross legs, and have a blond young lady stand behind your chair.

What an outsider regards as the gambler's superstitions are in fact manifestations of the gambler's hypothetical reconstruction of the world Fate has bit by bit revealed to him. Experience has taught him, say, that in order to win he must touch a hunchback on the day of play, carry a rabbit's foot in his left pocket, not sit at the gaming table with his legs crossed, and so on. This sort of knowledge is to him what, say, the knowledge of the mathematics of airflow over wings may be to an aircraft designer.

The programmer can convert every new embarrassment into a special case for a specially constructed, ad hoc subprogram.

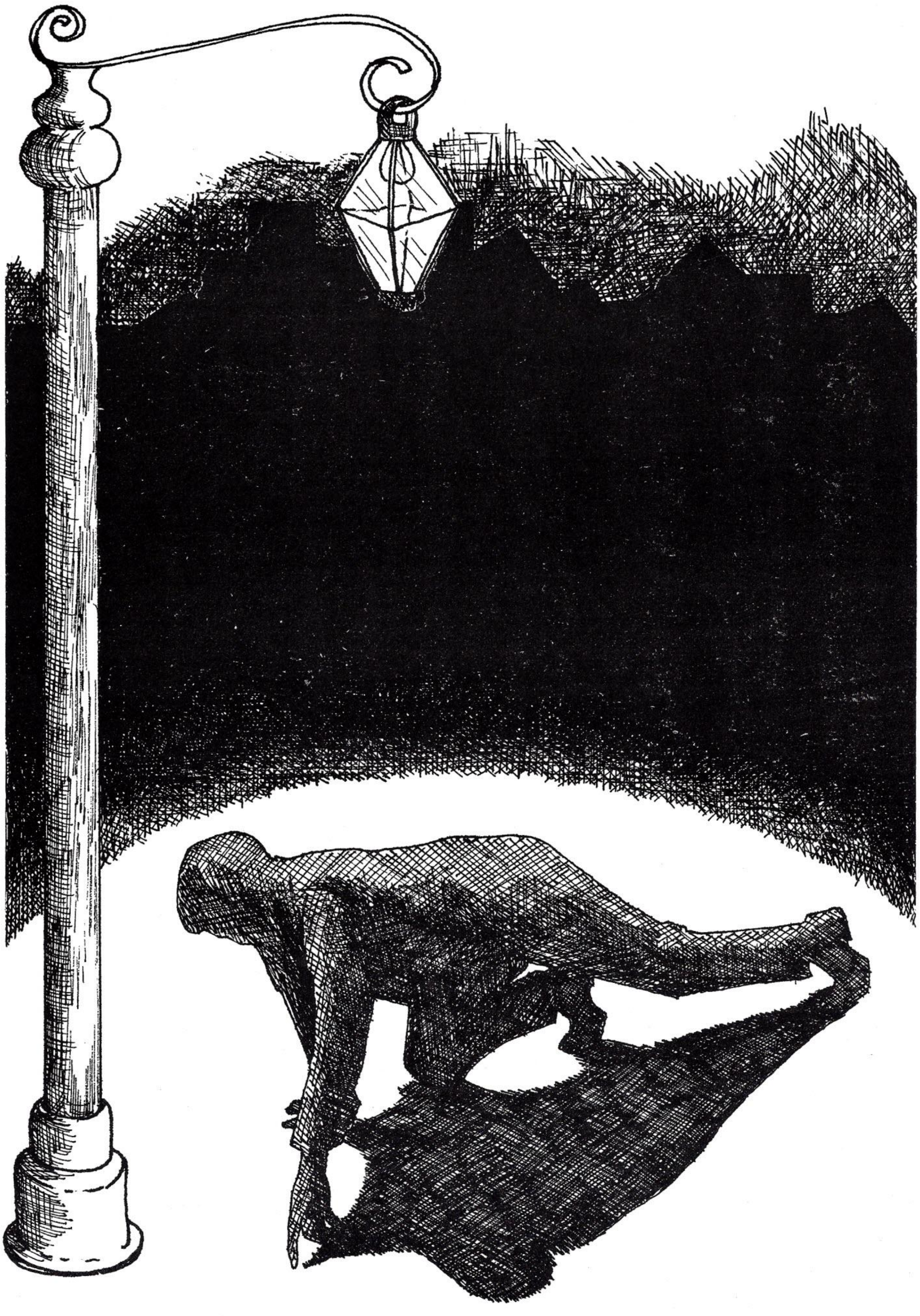
Because the gambler's superstitions are effectively irrelevant to the motions of dice, the orderings of cards, and so on, his hypotheses are very often empirically falsified. Each falsifying experience, however, contains certain elements that can be integrated into the main lines of his hypothetical framework and so save its over-all structure. Losing, therefore, doesn't mean that carrying a rabbit's foot, for example, is wrong or irrelevant, but only that some crucial ingredient for success has so far been overlooked. Perhaps the last time the gambler did win, a blond young lady stood behind his chair. Ah! So that's it: Touch a hunchback, carry a rabbit's foot, don't cross legs, *and* have a blond young lady stand behind the chair. When that doesn't work, he calculates that that particular combination works only on Thursdays, and so on and on and on. Bits and pieces of explanation are added on, some are removed, and the entire structure becomes more and more complicated. Eventually, the gambler really does command a conceptual framework that rivals a body of scientific knowledge, at least in its complexity and intricacy. He is an expert in a richly complicated world open only to the few initiates who have, through their own hard work and risktaking, learned its mysterious lore and language.

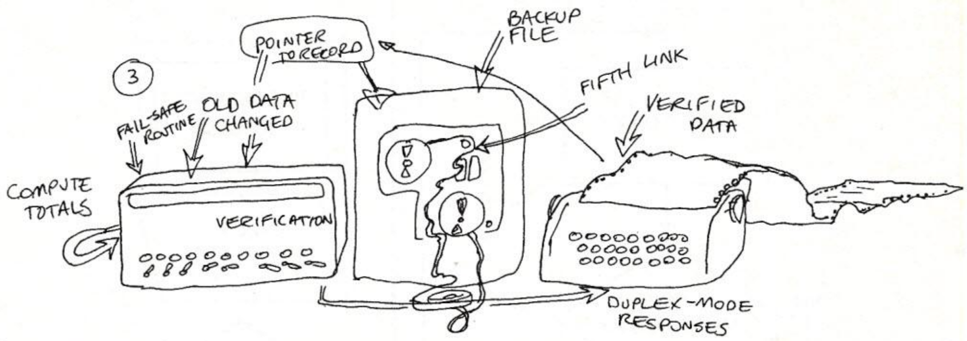
The magical world inhabited by the compulsive gambler is no different in principle from that in which others, equally driven by grandiose fantasies, attempt to realize their dreams of power. Astrology, for example, has constructed an enormously complex conceptual framework, a system of theories and hypotheses which allegedly permit the cognizant to control events. To know, for example, that the conjunction of certain planets on a particular date bodes ill for a particular venture, but that some other conjunction on some other date bodes well for it, and then to undertake that venture on the favored date, is to attempt to control events.

But the hypotheses of astrology too are routinely falsified by events. How then does astrology, and how do other such magical systems, remain at all a force in the minds of men? Exactly as do the hypotheses of the compulsive gambler. First, any contradiction between experience and one magical notion is explained by reference to other magical notions. Thus the entire structure of the magical system of beliefs is supported by its very circularity. This way of protecting the system against assaults by reality is especially effective if objections are always met one at a time, for then the very demonstration that an apparently anomalous fact can be incorporated into the system serves to validate the system. The gambler may, for example, appeal to the fact that he didn't tie his shoelace, as he knew he should have, to account for his "bad luck" on a particular day. That sort of explanation is formally equivalent to the compulsive programmer's assumption that his program's misbehavior is caused by a merely technical programming error.

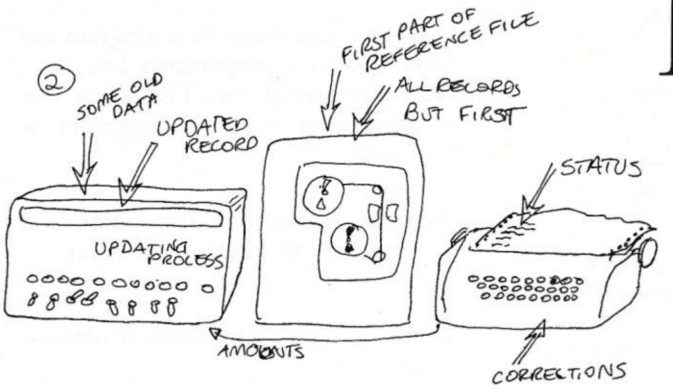
A second way in which conceptual frameworks of gamblers and of programmers are protected is by cyclical elaboration. The gambler who suddenly realizes that certain of his tricks work only on Thursdays simply incorporates this new "insight" into his already existing framework of superstitions, thus, in effect, adding an epicycle to its structure. The programmer is free to convert every new embarrassment into a special case to be handled by a specially constructed, ad hoc subprogram and to be thus incorporated into his over-all system. Such unbounded epicyclic elaborations of their systems provide both programmers and gamblers with an inexhaustible reserve of subsidiary explanations for even the gravest difficulties.

Finally, the conceptual stability of a magical or programming system may be protected by denying, to use the words of Michael Polanyi,




3
POINTER TO RECORD
BACKUP FILE
FIFTH LINK
VERIFIED DATA
FAIL-SAFE ROUTINE
OLD DATA CHANGED
COMPUTE TOTALS
VERIFICATION
DUPLEX-MODE RESPONSES

and Flow Right


2
SOME OLD DATA
UPDATED RECORD
FIRST PART OF REFERENCE FILE
ALL RECORDS BUT FIRST
STATUS
UPDATING PROCESS
AMOUNTS
CORRECTIONS

chart, although that wouldn't be a very descriptive flowchart. Alternatively, a box could represent one of the computer's elementary operations (like loading a number into the register, adding one number to another, or making a number negative, etc.).

The amount of detail you put into a box determines what sort of picture your flowchart will give. If you let one box represent a series of complicated instructions, the resulting flowchart will give you a rough, "thumbnail sketch" of the program. If each box represents the very simplest operation possible, the flowchart you create will give you a very detailed drawing, complete with the finest bits of information. Sometimes you can lose the picture for the detail, however, so it's common to generalize the flowchart a bit. In Figure 1, each box represents a very large and complex procedure. However, the flowchart still illustrates some of the basic structure of the program.

The sort of program arrangement illustrated in Figure 1 is called a *loop*—a term that reflects the way it looks in the flowchart. Like any good picture, this flowchart tells us more than just what we wrote down: we see that there's no way that this program can end! (That's very important to note, especially at this point in the development of a program. An "endless loop" can be particularly difficult to prevent—in fact, it has been proven that a computer cannot tell whether or not a program will enter such an endless loop. The burden of proving a program correct falls on you, the programmer.)

Since we obviously do want a program that will end (you could, I suppose, pull the plug instead), our next move should be to embellish the flowchart with some way of terminating the program. We do this with a new kind of box, the *diamond* or *decision-box.* A diamond always has one arrow (or flow-line) going into the top, and two or three lines leaving. Inside the box is written the test to be performed, and the flow-lines leaving the box are labeled with the conditions

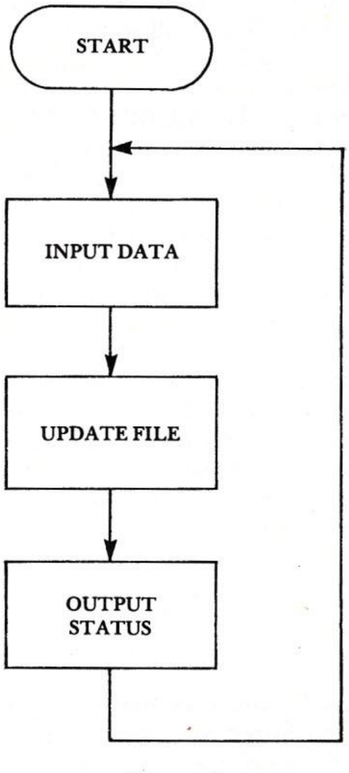

START
INPUT DATA
UPDATE FILE
OUTPUT STATUS

Figure 1.

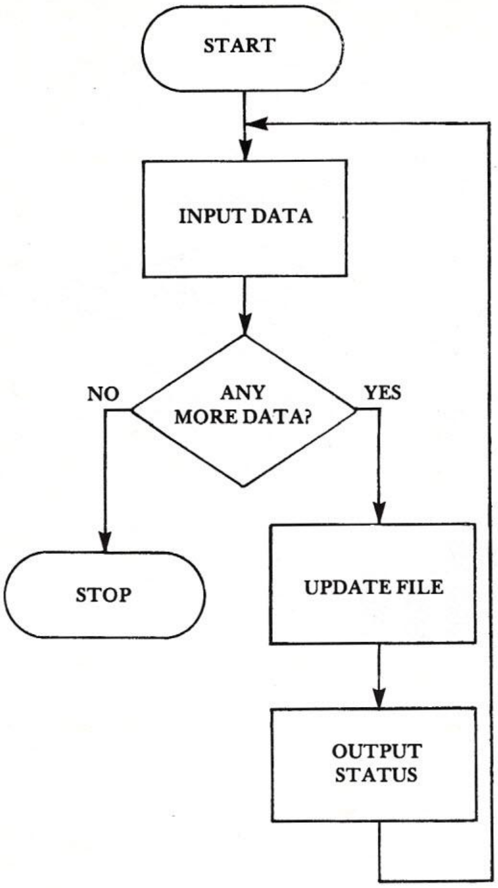
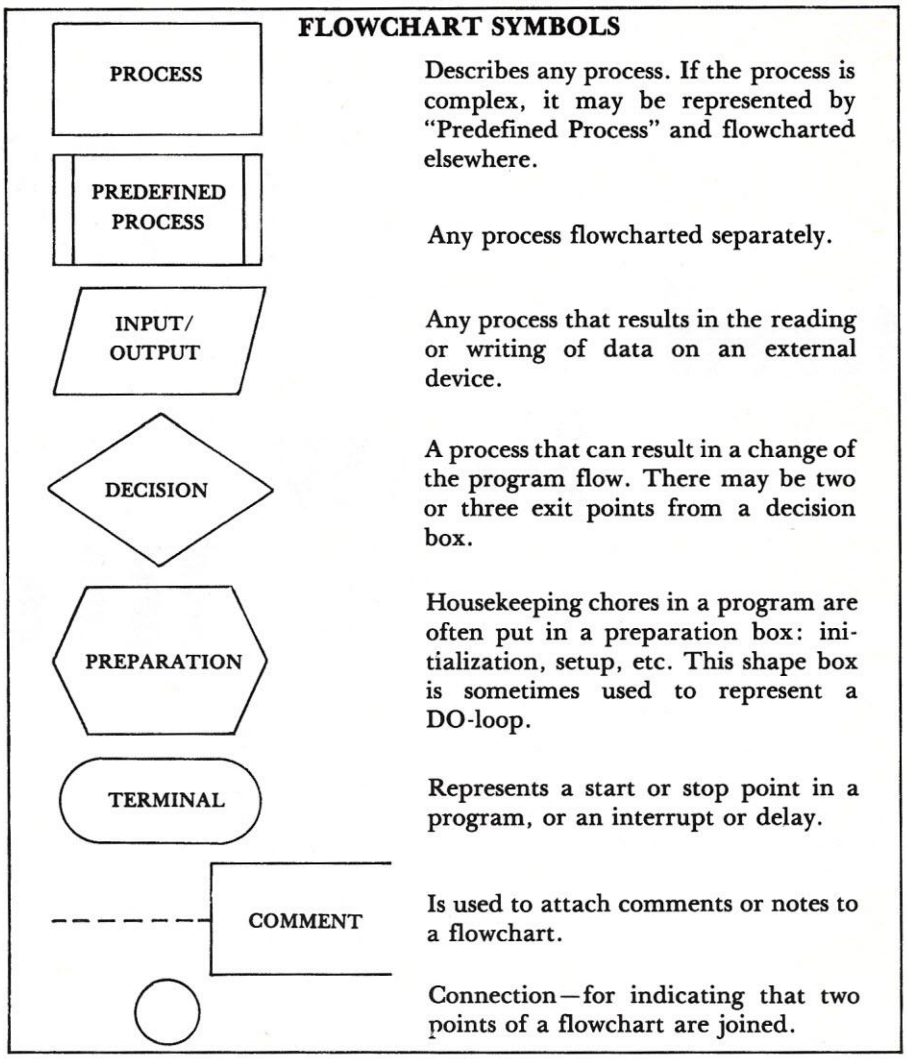


Figure 2.



FLOWCHART SYMBOLS

Describes any process. If the process is complex, it may be represented by "Predefined Process" and flowcharted elsewhere.

Any process flowcharted separately.

Any process that results in the reading or writing of data on an external device.

A process that can result in a change of the program flow. There may be two or three exit points from a decision box.

Housekeeping chores in a program are often put in a preparation box: initialization, setup, etc. This shape box is sometimes used to represent a DO-loop.

Represents a start or stop point in a program, or an interrupt or delay.

Is used to attach comments or notes to a flowchart.

Connection—for indicating that two points of a flowchart are joined.

under which each line would be followed. This is illustrated in the flowchart in Figure 2.

Here we've added the decision box labeled "ANY MORE DATA?" If there wasn't any more—presumably the user signalled this when the program was asking him to INPUT DATA in the rectangle above—then the program quits. Otherwise, the program continues with updating the file, and so on.

Still, the flowchart in Figure 2 also is pretty coarse: it would be hard to write a program directly from the information in this flowchart. What we need to do is break the boxes into the components that they represent. For example, the box labeled UPDATE FILE represents quite a few operations. If we assume that the file referred to is a direct-access file (as on a floppy disk), the steps necessary to update the file would include reading the old record to be updated (probably referenced by part number or something of the sort), writing an error message if the record didn't exist, modifying the record, and rewriting it. The flowchart could be expanded to look like Figure 3.

Now the flowchart contains enough information to give the programmer a good idea of what's involved in the program. The parallelogram boxes represent input-output operations (although you could certainly use rectangular boxes instead). The other square boxes could be broken up into their component pieces, if desired, but they tell a reasonable amount of information as it is.

Flowcharts are part of that all-too-neglected side of programming, documentation. As any programmer will tell you, the most important thing you can give someone with a program is documentation on how it works and why it was done that way. Any programmer will tell you, too, that documentation is the biggest pain since hexadecimal arithmetic.

Never mind that, it's critical to the success of a program that a lot of information about it be available. Most computer languages contain a mechanism for inserting comments on a program's operation among the statements. BASIC uses the keyword REM ("remark"), FORTRAN uses a C ("comment"), IBM 360 assembler uses an asterisk, and APL uses a picture of a lightbulb (get the idea?) to prefix a comment. COBOL, on the other hand, tries to be self-documenting, in that it uses wordy English or English-like expressions to describe what it does.

All these methods are handy, but a REM in BASIC is likely to be followed by something like "This pro-

The new technique—structured programming—is quite the rage in some places.

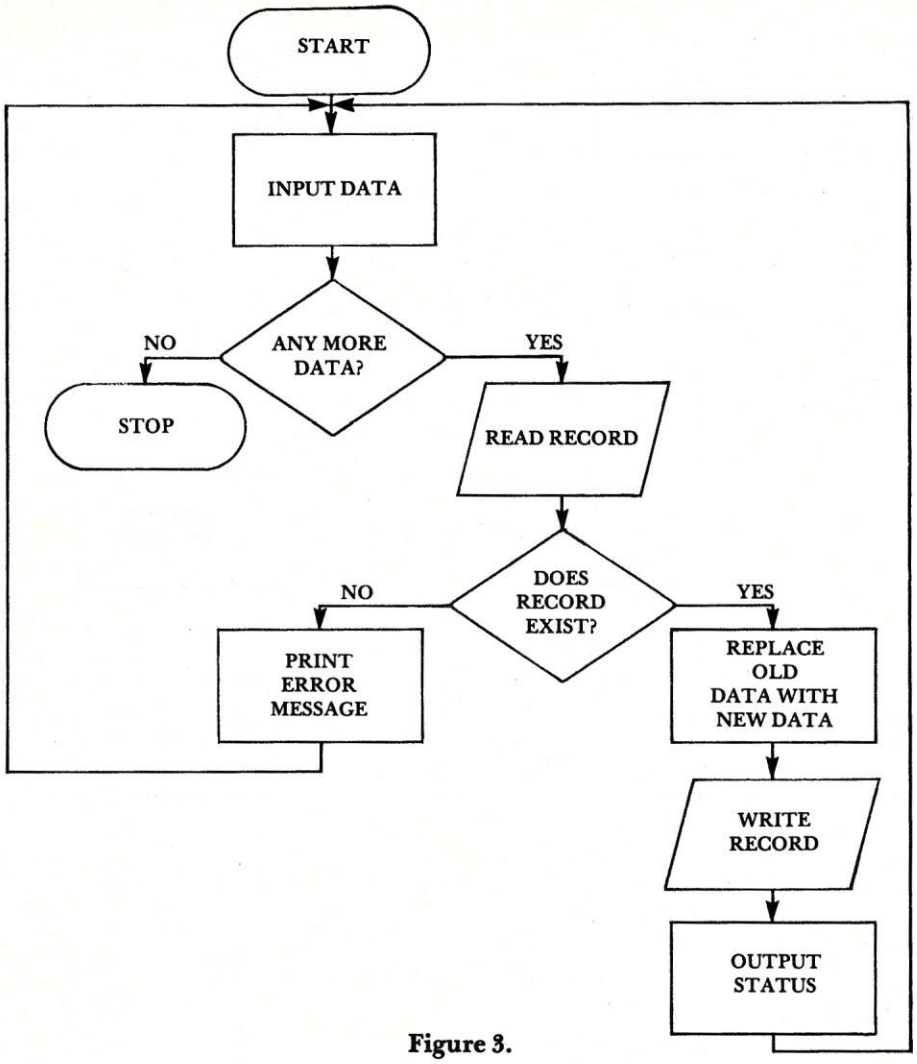


Figure 3.

gram was written by John Smith. . . ." rather than a detailed description of what the program does. Besides, the program itself constitutes a description of what it does. None of these methods, by themselves, give a good picture of what a program does. Hence, non-verbal methods (flowcharts) are used constantly.

In fact, programming languages today parallel flowcharts for a very good reason: most languages were developed from flowcharting methods. Now, new pictures of programs are leading to new languages, and vice versa. Take for example the humble loop. The simplest form of a loop uses a counter to number the repetitions of the loop, and it's set up in flowchart form as shown in Figure 4.

Tree diagrams are not dependent on the linear nature of the computer.

If the END-VALUE is 12, this loop executes what's in the shaded box twelve times. This construction is so common that most widely-used languages have a construction just for it, called the DO or FOR loop.

Usually the three unshaded boxes shown in Figure 4 are coded as one programming language statement. But a flowchart really should have all three components included, because of the way they bracket the instructions to be executed. This has led some people to construct a new box for their flowcharts, one that's often represented as a hexagon, with various arrows leading in and out of it. It's led other people to reject the notion of the three-part loop setup entirely, and think of loops as un-flowchart-like.

In fact, this attitude is being extended to the point where some new languages are without this type of flowchart structure entirely, and are written with all program flow changes done by a series of nested loops. The new technique is called structured programming, and it's quite the rage in some places. The idea is that the structure of the programming language forces the programmer to write in a certain style; therefore other programmers should find it understandable. It's also impossible to flowchart.

Since it is very important to get pictures of whatever's going on (for me it's important, anyway), other sorts of pictures have been invented. One of them is called a HIPO chart, and it consists of breakdowns of the data flow—what values are being passed to what subroutines and what's being returned. In doing this, it also gives a picture of the program flow: what pieces of the program are being executed and when, but mostly in the context of the values they return.

Another variant on this idea is a tree diagram, with the base at the top of the page. Fanning down are subroutines, presumed to handle one little task of the program's execution. Unlike a family tree (it looks like one), one subroutine can be in any number of places. The detail at any level of the

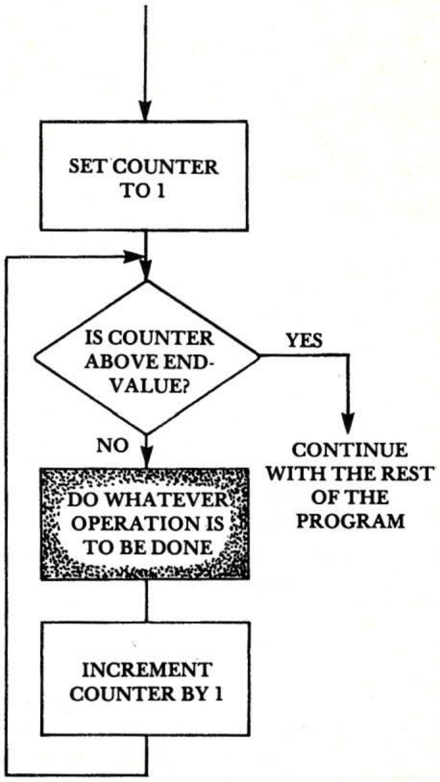

SET COUNTER TO 1
IS COUNTER ABOVE END-VALUE?
YES
NO
CONTINUE WITH THE REST OF THE PROGRAM
DO WHATEVER OPERATION IS TO BE DONE
INCREMENT COUNTER BY 1

Figure 4.

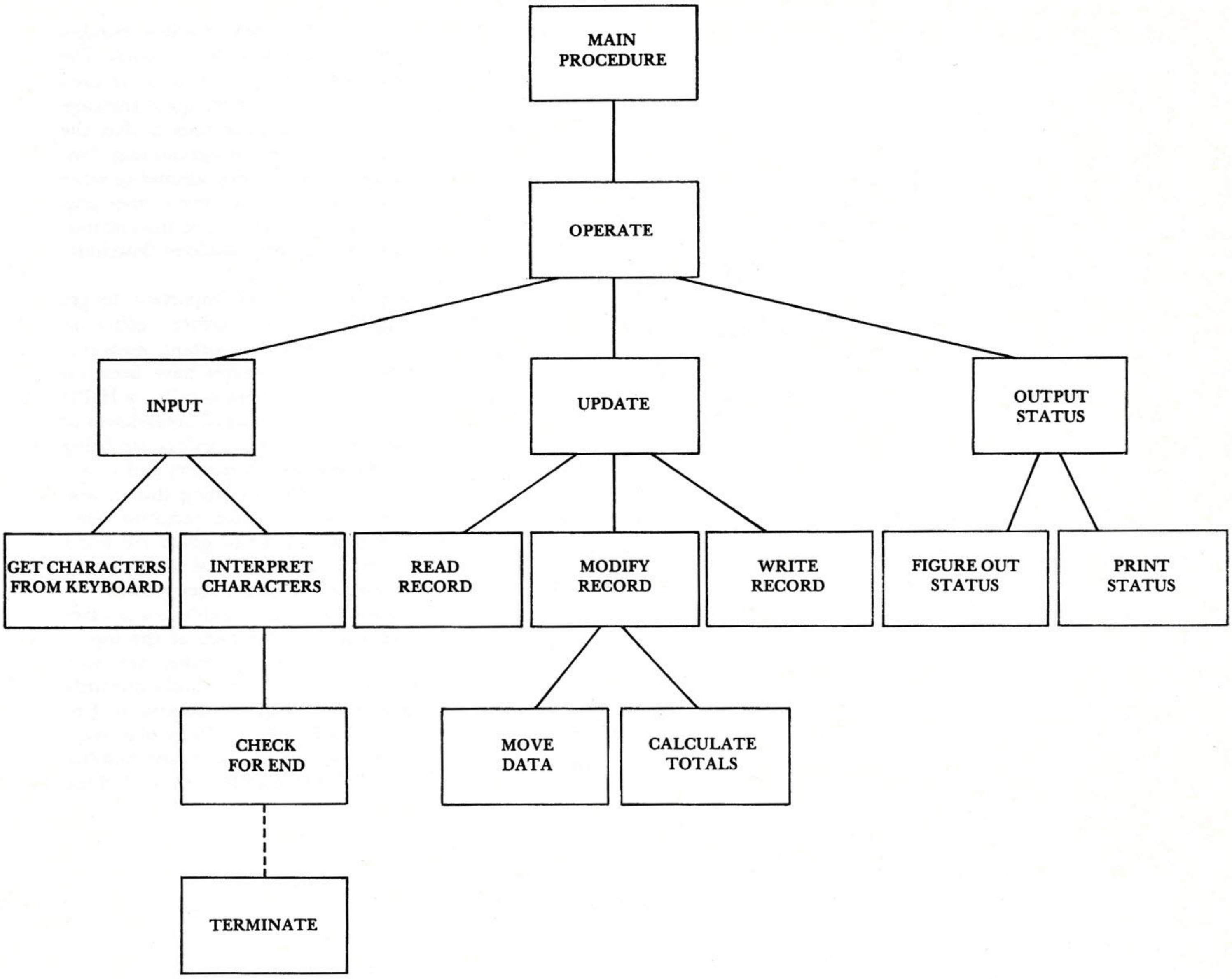

MAIN PROCEDURE
OPERATE
INPUT
UPDATE
OUTPUT STATUS
GET CHARACTERS FROM KEYBOARD
INTERPRET CHARACTERS
READ RECORD
MODIFY RECORD
WRITE RECORD
FIGURE OUT STATUS
PRINT STATUS
CHECK FOR END
MOVE DATA
CALCULATE TOTALS
TERMINATE

Figure 5.

tree corresponds to the detail of the subroutine that handles that part. Program flow within a routine is indicated by the left-to-right order of the subroutines that branch off.

If you want to know how to handle loops, it's easy: one subroutine level represents the looping mechanism, and the one below it represents the thing that gets executed. Subroutines that get called from a number of places in the program might be written down once, and the name of the procedure would be inserted in the graph of the main program. The graph for the data file program which I flowcharted earlier could look like Figure 5 in a tree diagram. A number of other versions are possible, too.

What does this get you? Unlike flowcharting, the tree diagram method of describing, graphically, what a program does is not dependent on the linear nature of the computer. But it is tied to another concept—that of subroutines and "subroutining"—so closely that it would not be useful with a language that doesn't allow subroutines. (Luckily, there aren't many like that.)

Tree diagramming also binds the programmer to a style of programming, which I'd rather not do. Any good programmer has a real "style" of his own, which is probably most effective for his needs.

Somewhere there's probably an ideal scheme for illustrating a problem-solving procedure, or program, one that implies nothing about "good" programming style, and that can be used with any language. I don't know what it is, though, and so I use my own: I draw every kind of diagram I can think of, write my program from these, and flowchart after I'm done if someone tells me to.

But some time I'm going to give them all up, and develop the perfect scheme. It's going to be a series of drawings, like an animated movie, a frame for the computer, for the teletype, and for the disk drive. All you'll have to do is label them with what goes on at each moment. Play it as it lays and you'll get the whole picture—perfect every time. ▼

Small-Business Payroll Program Follow-Up

by Robert G. Forbes

For those of you interested in further documentation on the payroll program in the last issue of *ROM,* here are the module definitions in detail, plus the flowcharts.

The module definitions are as follows:

File Maintenance (EMP.1)

A. Add—Add a new employee.
B. Change—Change a field within the employee file.
C. Examine—Examine employee file.
D. List—List employees and employee's number.
E. Return—Return to the operating system.
F. Emp. 2—Execute payroll calculations.

Looking through the program, we can see that the major file is opened as a random file. (This is to allow us to access a given record with greater speed and ease.) STMT numbers 100-276 ask for one of the selections and then process the request.

The selection "EXAMINE" allows the operator to print the employee file on either the printer or the CRT.

STMT numbers 290-440 describe what the employee file actually looks like.

STMT numbers 600-760 allow the operator to correct individual fields within an employee record.

STMT numbers 800-910 allow the operator to print the employee master file out on the CRT.

STMT numbers 1070-1230 print the EMPMST file out on the CRT.

Calculation Module (EMP.2)

This is the nucleus of the payroll system, as it is here that the actual payroll is calculated and both the EMP.YTD and EMP.QTD files are updated. If you look at the flowchart for the program, you can see the first thing we do is to build the tax tables. There are four major tables:

1. Single weekly variables (S1 (*x*), S2 (*x*), S3 (*x*), S4 (*x*))
2. Married weekly variables (M1 (*x*), M2 (*x*), M3 (*x*), M4 (*x*))
3. Single monthly variables (H1 (*x*), H2 (*x*), H3 (*x*), H4 (*x*))
4. Married monthly variables (G1 (*x*), G2 (*x*), G3 (*x*), G4 (*x*))

After loading the tables, the program starts reading in the employee master file which we maintained in EMP.1. Once we have determined if an employee is single or married, and is paid weekly or monthly, the actual calculations take place and are printed out in STMTs 610-690. The detailed figures are stored in memory for use later.

The program then does the same procedure for the rest of the employees. After all of the employees have been processed, the employer receives a summary of what the detailed pay and deductions were for each one of the employees, as well as totals for the entire group.

Once the program prints the summary report, it updates both the EMP.YTD and EMP.QTD files, and then links to the program EMP.3.

Employee YTD Module (EMP.3)

This module reads the employee YTD file and prints a report listing YTDs of Social Security numbers, gross, Ded. tax, FICA, and net of all employees. It links to EMP.4.

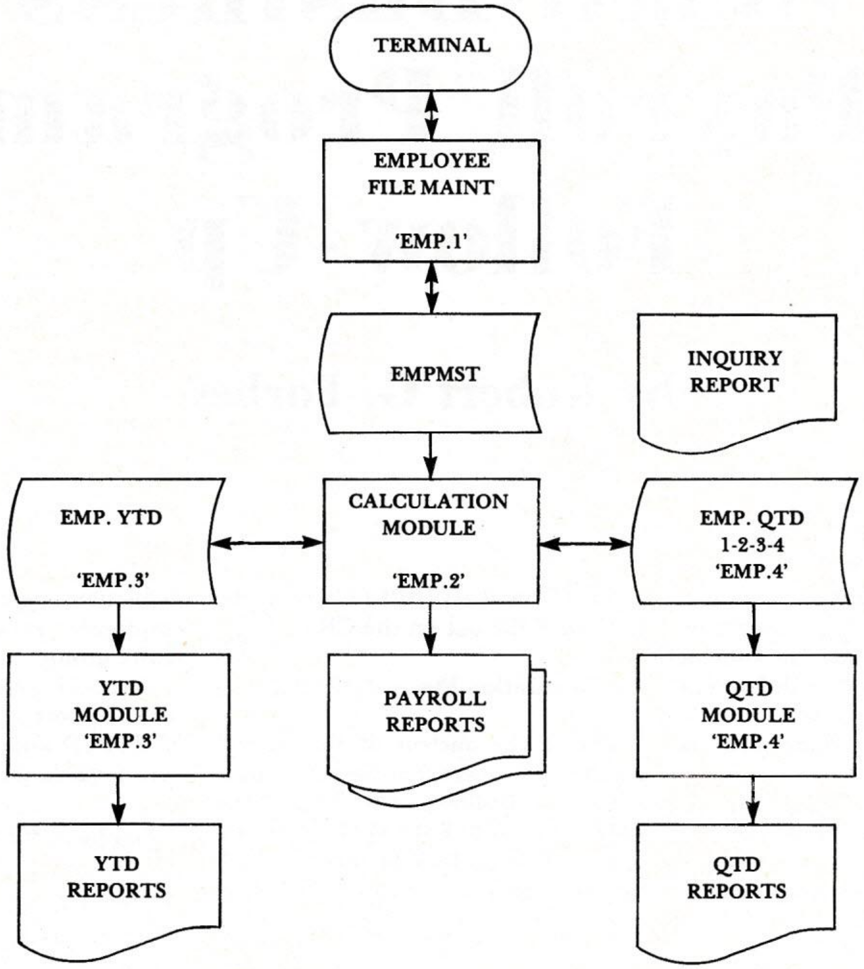
Employee QTD Module (EMP.4)

This module is the same as EMP. 3, except that it prints out the quarter to date figures.

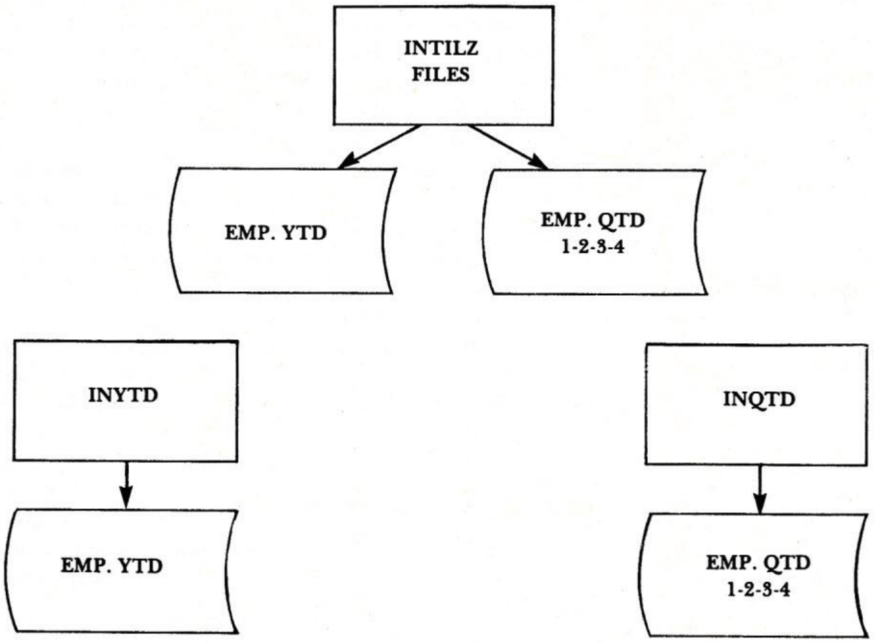
The system described in the article in last month's *ROM* is the core of a payroll-personnel system. The personnel phase of the system would include the extraction of statistical data pertinent to the particular type of business of the user, as well as listings of personnel history reports and various exception reports.

In addition to the modules discussed here, a backup program should be used to copy the data file. By doing this, one has added security in case some accident (loss, fire, or damage) happens to the original. One could then use the copied version. Systems do crash and backup records are a key component of any payroll system and make it worth its weight in IRS forms. ▼

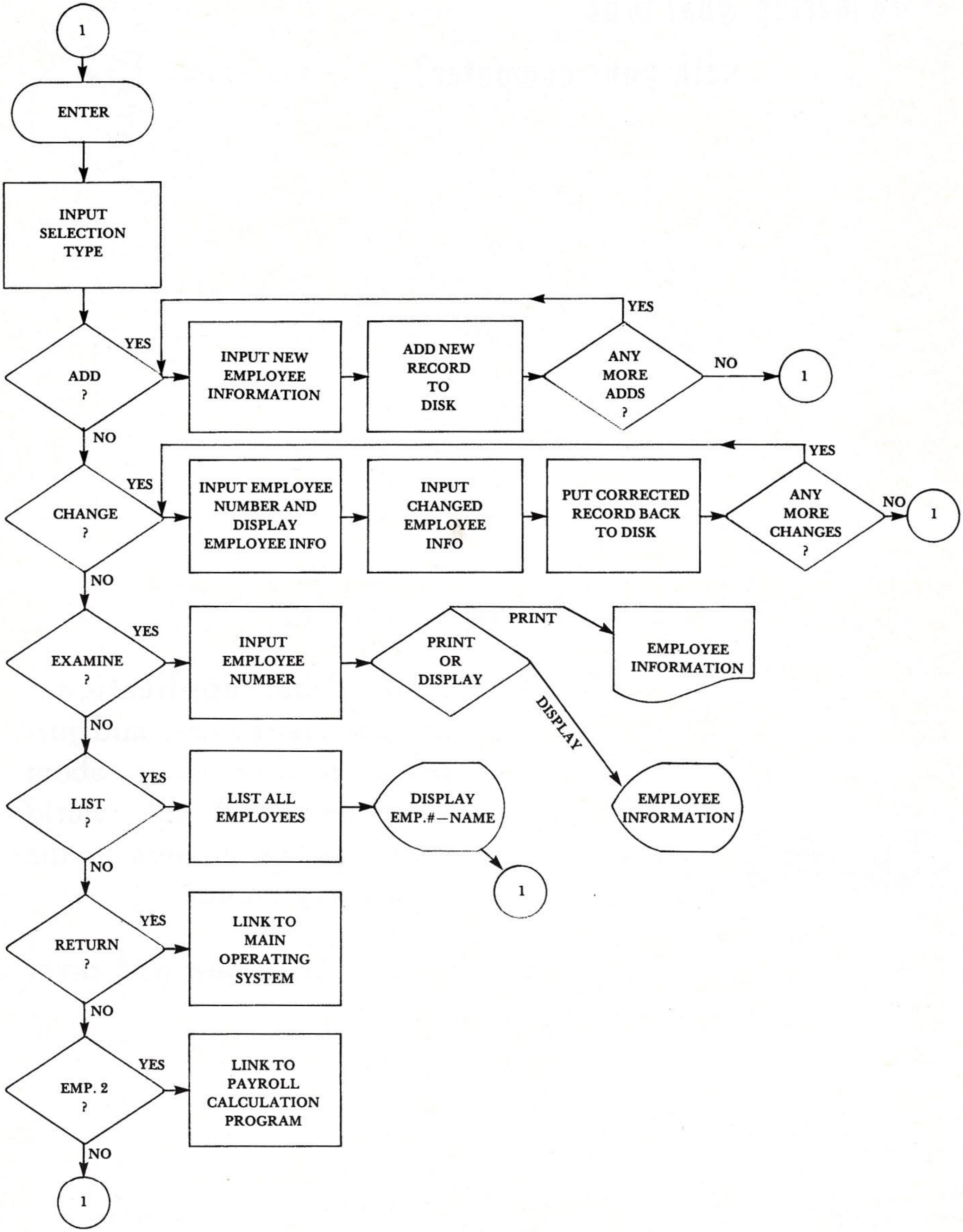
PAYROLL SYSTEM OVERVIEW


TERMINAL
EMPLOYEE FILE MAINT
'EMP.1'
EMPMST
INQUIRY REPORT
EMP. YTD
'EMP.3'
CALCULATION MODULE
'EMP.2'
EMP. QTD 1-2-3-4
'EMP.4'
YTD MODULE
'EMP.3'
PAYROLL REPORTS
QTD MODULE
'EMP.4'
YTD REPORTS
QTD REPORTS

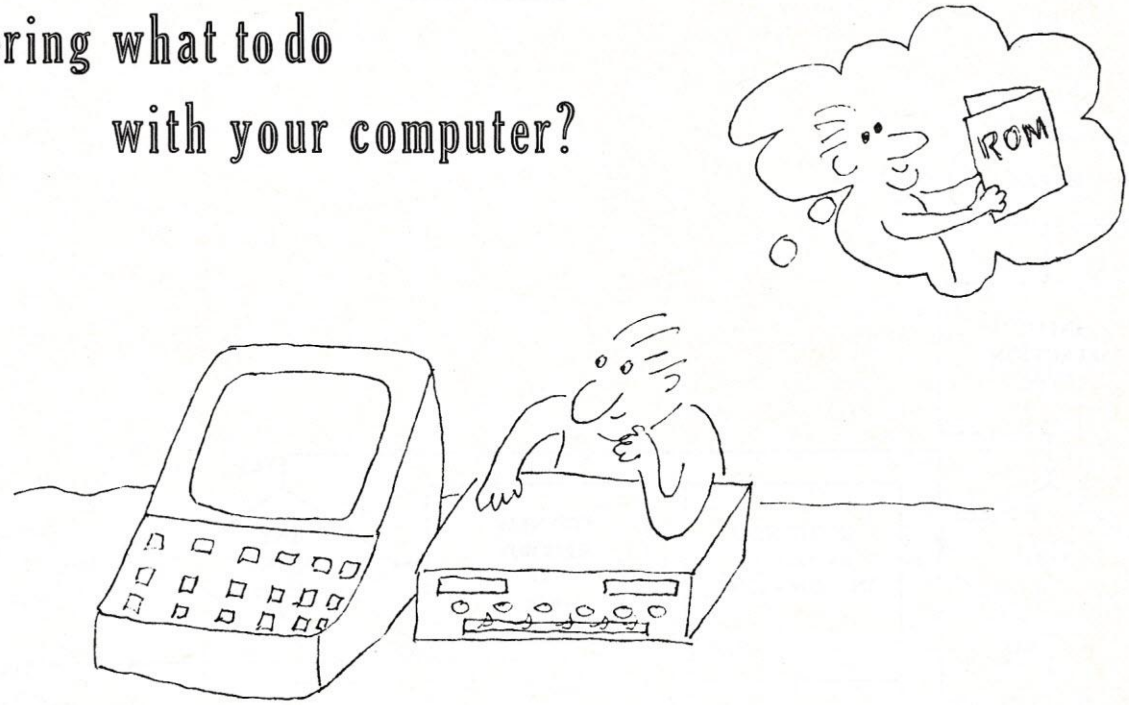
PAYROLL INITIALIZATION OF FILES SYSTEM OVERVIEW


INTILZ FILES
EMP. YTD
EMP. QTD 1-2-3-4
INYTD
INQTD
EMP. YTD
EMP. QTD 1-2-3-4

EMP.1
FILE MAINT


1
ENTER
INPUT SELECTION TYPE
ADD ?
YES
NO
INPUT NEW EMPLOYEE INFORMATION
ADD NEW RECORD TO DISK
ANY MORE ADDS ?
YES
NO
1
CHANGE ?
YES
NO
INPUT EMPLOYEE NUMBER AND DISPLAY EMPLOYEE INFO
INPUT CHANGED EMPLOYEE INFO
PUT CORRECTED RECORD BACK TO DISK
ANY MORE CHANGES ?
YES
NO
1
EXAMINE ?
YES
NO
INPUT EMPLOYEE NUMBER
PRINT OR DISPLAY
PRINT
EMPLOYEE INFORMATION
DISPLAY
EMPLOYEE INFORMATION
LIST ?
YES
NO
LIST ALL EMPLOYEES
DISPLAY EMP.#—NAME
1
RETURN ?
YES
NO
LINK TO MAIN OPERATING SYSTEM
EMP. 2 ?
YES
NO
LINK TO PAYROLL CALCULATION PROGRAM
1

Wondering what to do with your computer?



ROM will show you.

New ideas, applications, peripherals, games, and just plain good reading about computers and the world from the best writers in the field every month.

Subscribe now and save.

ROM
COMPUTER APPLICATIONS FOR LIVING

ROM Publications Corp.
Route 97
Hampton, CT 06247

Name ______________________

Address ______________________

City ______________________

State __________ Zip __________

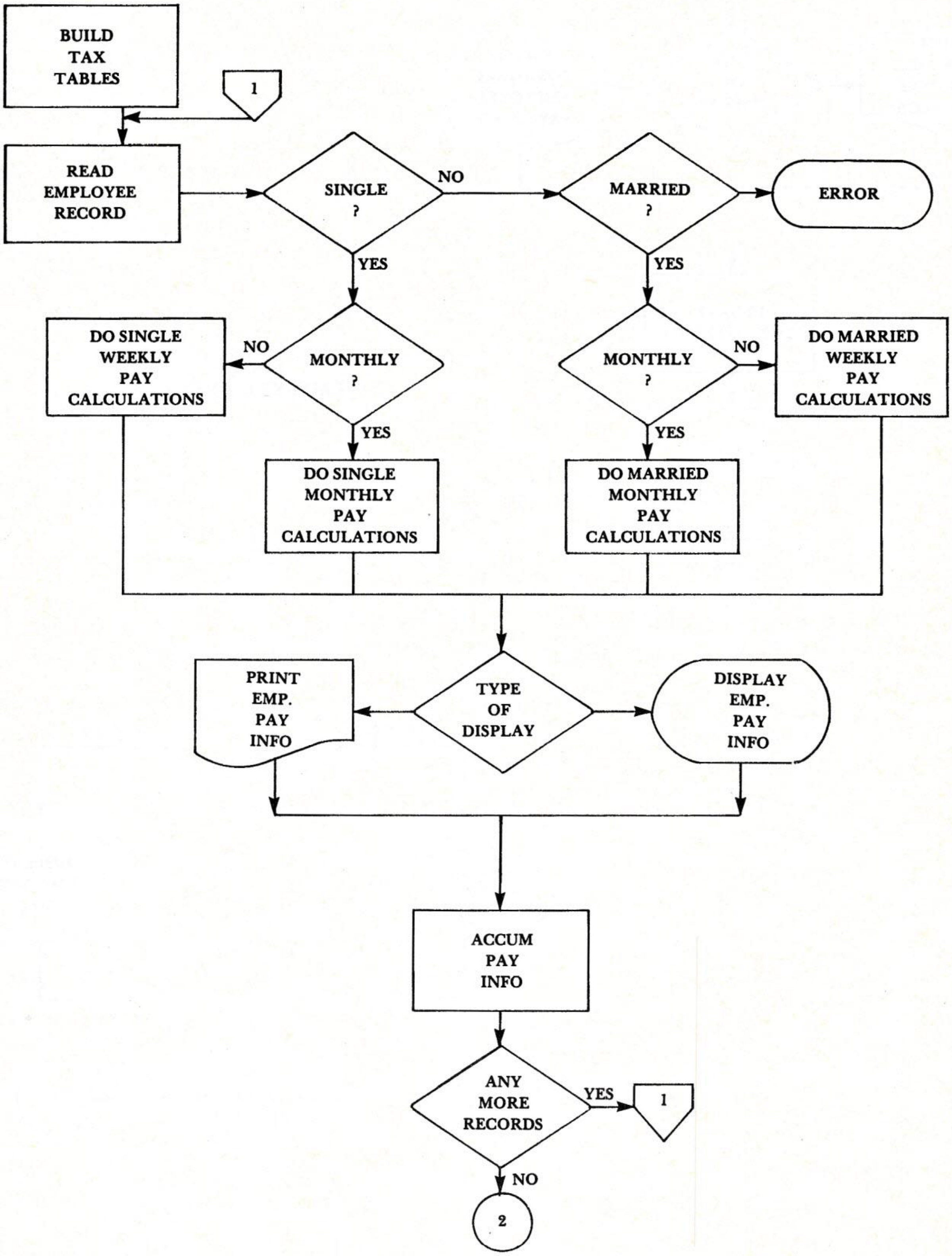
☐One year $15 ☐Two years $28 ☐Three years $39
☐Check or money order enclosed.
☐Master Charge ☐BankAmericard
Exp. date __________ Card# __________

° Please allow 4-6 weeks for delivery.

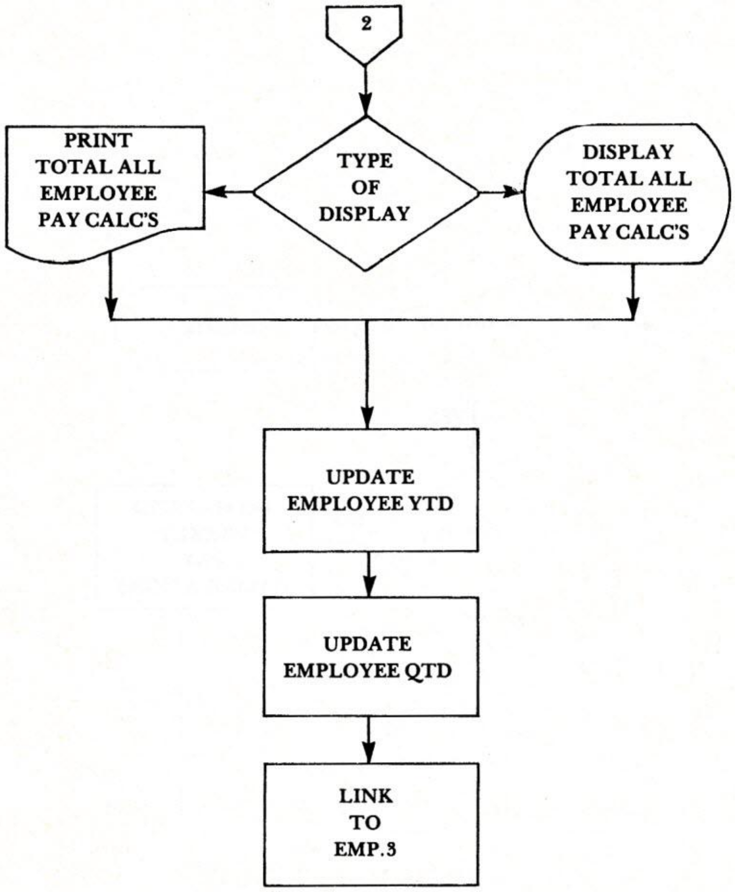
GUARANTEE

If not satisfied with **ROM** at any time, let us know. We'll cancel your subscription and mail you a full refund on all copies still due you.

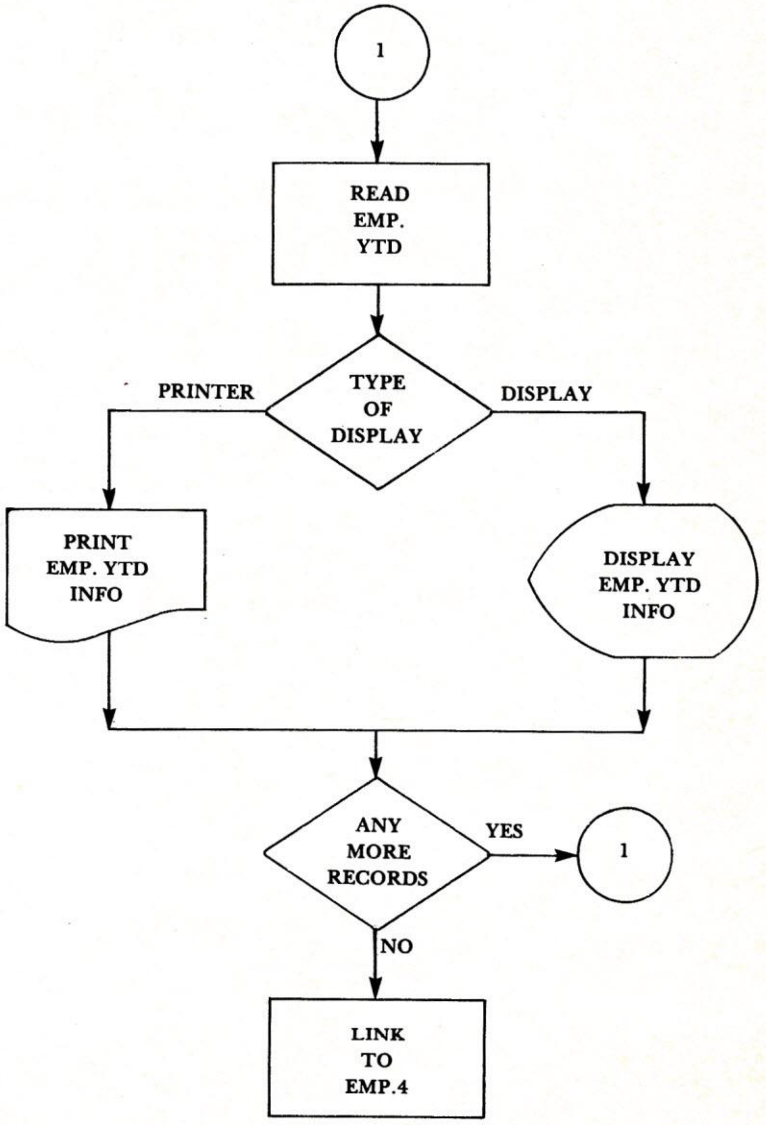
EMP.2
PAY CALCULATIONS


BUILD TAX TABLES
1
READ EMPLOYEE RECORD
SINGLE ?
NO
MARRIED ?
ERROR
YES
YES
DO SINGLE WEEKLY PAY CALCULATIONS
NO
MONTHLY ?
MONTHLY ?
NO
DO MARRIED WEEKLY PAY CALCULATIONS
YES
YES
DO SINGLE MONTHLY PAY CALCULATIONS
DO MARRIED MONTHLY PAY CALCULATIONS
PRINT EMP. PAY INFO
TYPE OF DISPLAY
DISPLAY EMP. PAY INFO
ACCUM PAY INFO
ANY MORE RECORDS
YES
1
NO
2

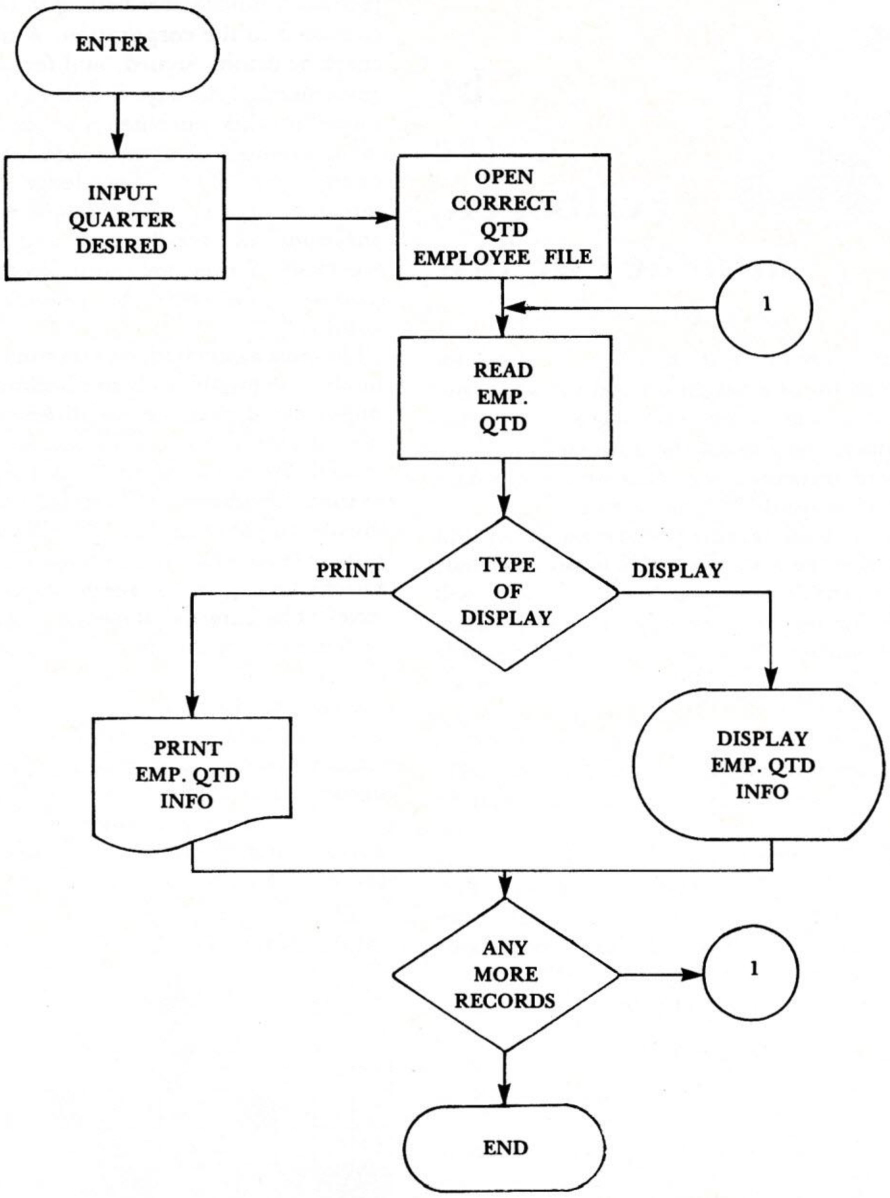
EMP.2
PAY CALCULATIONS


2
PRINT TOTAL ALL EMPLOYEE PAY CALC'S
TYPE OF DISPLAY
DISPLAY TOTAL ALL EMPLOYEE PAY CALC'S
UPDATE EMPLOYEE YTD
UPDATE EMPLOYEE QTD
LINK TO EMP.3

EMP.3
EMP. YTD PRINT


1
READ EMP. YTD
PRINTER
TYPE OF DISPLAY
DISPLAY
PRINT EMP. YTD INFO
DISPLAY EMP. YTD INFO
ANY MORE RECORDS
YES
1
NO
LINK TO EMP.4

EMP.4
EMP. QTD REPORT


ENTER
INPUT QUARTER DESIRED
OPEN CORRECT QTD EMPLOYEE FILE
1
READ EMP. QTD
TYPE OF DISPLAY
PRINT
DISPLAY
PRINT EMP. QTD INFO
DISPLAY EMP. QTD INFO
ANY MORE RECORDS
1
END

Legal ROMifications

YOU, INC.?

by Peter Feilbogen, attorney at law

Whether you're contemplating starting a computer store, setting yourself up as a neighborhood software consultant, or, for that matter, manufacturing a few specialized boards—no matter how small the business, you've no doubt thought about incorporating. And well you should have. But exactly what would this move mean?

A corporation is an artificial entity whose authority and powers are derived from the State in which it is incorporated. For a small business, the State in which a corporation will do business is fine for its incorporation. It is only when there will be large numbers of stockholders, as in a public corporation, that it is necessary to choose the State of incorporation with great care.

The word *entity* is very important in this definition. A corporation is separate and distinct from its directors, officers, and employees. In many states a corporation may be structured with one man being the sole incorporator, director, and officer. This distinction as a separate entity must be reflected in the corporate books and records. Personal transactions must be kept separate and apart from those of the corporation. Probably the most celebrated feature of a corporation, that of limited liability, has its foundation in the separate entity concept.

The concept of *limited liability* acts as a great magnet in drawing businessmen to the corporate mode. The corporation has assets and capital, however modest, and liabilities. The creditors may look only to the assets and capital of the corporation to satisfy their claims. The personal assets of directors, officers, employees, and stockholders are safe from the claims of creditors.

The corporate structure limits the risk which individuals incur in starting and operating a business. That is certainly terrific, but it is an open secret. Suppliers and lending institutions are well aware of it. Just as one businessman tries to avert personal responsibility, the other businessman tries to fix such liability. The most common methods are by requiring cash with order, requiring payment within short periods of time, i.e. five to ten days, insisting on personal guarantees, or even refusing to deal with the corporation.

Limited liability is also extremely important in dealing with law suits. Assume an employee making a business trip in a corporate vehicle strikes and kills a person who earns $100,000 a year and has a wife and two young children. When the corporation is served with a multi-million dollar law suit, well in excess of the insurance carried, only the assets of the corporation are in jeopardy. The principal of the corporation is able to safeguard his personal assets.

So far this looks perfect, but there must be a flaw. Indeed, there is. The most serious exception to limited liability lies in the "trust fund" area. Businesses collect various taxes which must be transmitted to governmental authorities. The most common are payroll deductions and sales tax. Such funds never belong to the corporation; they are entrusted to the corporation. A person must direct that a check be drawn, signed, and forwarded to the appropriate governmental agency. It follows that the diversion of such funds to other purposes is a personal liability. This personal liability falls upon a wide range of people, depending on the extent of their knowledge and authority. This point should be kept in mind by those who are employed by corporations in accounting and managerial functions, especially if they are authorized check signatories. Remember, you could be personally responsible for this liability.

The costs associated with starting a corporation are quite modest, approximately five hundred dollars. What is more important is that the tax differential between a corporation, a partnership, and a sole proprietorship can be controlled. Since a corporation is a separate entity, it is taxed as such. Dividends are paid from after-tax profits and are income to the recipients. The result is called double taxation by those who are affected. For some time small corporations having ten or fewer shareholders have been given relief. The Internal Revenue Code has allowed such corporations to elect to be taxed as partnerships. Such corporations are called "Subchapter S" corporations in honor of the Internal Revenue Code section of that name. The result is to confer on certain small corporations—the size usually involved in garage-type manufacturing operations producing computer hardware, or the size of small software shops and retail stores—the benefits of being incorporated and the relief from the double taxation burden, the best of all possible worlds. ▼

Solution to last month's PROMpuzzle

D	A	T	U	M	■	D	I	O	D	E	■	F	A	R	A	D
O	B	E	S	E	■	I	L	I	A	D	■	A	D	O	B	E
N	A	■	A	S	■	G	E	L	■	I	N	D	O	■	A	L
O	T	U	■	S	A	I	S	■	A	T	E	E	■	E	S	T
R	E	S	T	A	R	T	■	A	L	O	E	■	P	L	E	A
■	■	■	A	G	E	■	A	M	E	R	■	V	I	A	■	■
C	Y	C	L	E	■	A	L	P	S	■	S	I	G	N	A	L
R	O	P	E	■	E	M	I	L	■	S	O	P	S	■	D	O
I	K	E	■	S	T	A	T	I	C	M	O	S	■	A	D	A
M	E	■	M	A	T	T	■	F	L	I	T	■	B	R	E	D
E	S	C	A	P	E	■	F	I	A	T	■	N	E	A	R	S
■	■	A	S	S	■	D	I	E	T	■	M	E	L	■	■	■
P	E	R	K	■	P	I	E	R	■	L	O	W	L	O	S	S
R	A	D	■	C	E	R	F	■	M	I	S	S	■	P	A	W
O	S	■	S	A	N	E	■	P	A	N	■	Y	R	■	V	I
M	E	E	T	S	■	C	L	O	C	K	■	N	A	R	E	S
S	L	I	C	E	■	T	A	P	E	S	■	C	H	E	S	S

IT TOOK THE MEDIA TO KEEP THE GOVERNMENT HONEST. BUT IT TAKES "MORE" TO KEEP THE MEDIA HONEST.

MORE watches the media while the media watches you

What if the news reporters, TV commentators, gossip columnists and media gurus who helped write the Watergate story did as thorough a job investigating their own business?

What if Woodward and Bernstein found a Deep Throat somewhere in the bowels of the Times?

Or if Barbara Walters put Barbara Walters under the microscope?

That's the kind of thing that happens each month in the pages of MORE, the Media Magazine.

MORE covers the media like the media itself covers a big story. By looking for sources and listening and digging and watching every word and reading between the lines. By getting behind the scenes and into the back rooms and conference rooms, providing the stories behind the stories you get and the stories behind the stories you never get.

At MORE, we get media people to tell us things they'd never tell anyone else. And we get the very people who report, comment and advertise to write things about reporting, commentating and advertising they could never write anywhere else.

For example we've explored a family feud at the *Times* that may have been responsible for Daniel P. Moynihan becoming a U.S. Senator. We examined the power of a handful of editors at *Time* and *Newsweek* to create and destroy rock stars overnight. We interviewed the controversial *Los Angeles Times* reporter who said that journalists should "lie, cheat, steal or bribe to get their story." And MORE ran the Nora Ephron media column that *Esquire* killed and the profile of Rupert Murdoch that *New York* wouldn't run.

We've given the business to the news business, advertising, movies, publishing and the entire communications industry. And don't think they haven't started watching their words a little more closely now that they know someone else is.

So if you subscribe to the idea that someone should be watching the media like the media watches everyone else, subscribe to MORE.

I WANT TO KEEP THE MEDIA HONEST. SEND ME MorE.

Please enter my subscription immediately.

☐ $12 for 1 year (12 issues)
☐ $21 for 2 years (24 issues)
☐ $30 for 3 years (36 issues)

☐ Payment Enclosed ☐ Bill me

Name ____________________

Address ____________________

City __________ State __________ Zip __________

Signature ____________________

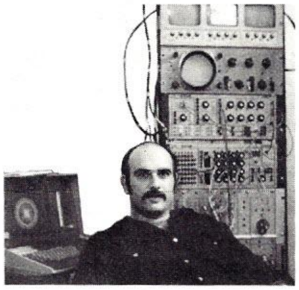
MorE THE MEDIA MAGAZINE P.O. Box 955 Farmingdale, New York 11735

MR57

Add $2.00 per year for outside U.S. and Canada. Please allow up to 4 to 6 weeks for delivery of first issue.

futuROMa

PLAYING THE PERCENTAGES



by Bill Etra

One of the basic problems with speeding up information processing within today's computer, be it micro or macro, is the limitation of the electronics themselves. We're theoretically feeding electrons through the circuitry at the speed of light through a vacuum. In reality, it's slower. Still a lot of the current emitter-coupled logic is coming awfully close to that limit already. So where do we go from here?

We can shrink the hardware even more than today's micro-miniaturized chips. IBM is already writing circuits with electron beams. Which means not only are we getting very close to the theoretical limits for speed, but the probable limits for the actual digital circuitry as well. So where else do we go?

Well, we can parallel-process a lot of information, and make larger machines. Sixteen-bit words, thirty-two-bit words, sixty-four-bit words, hundred-bit words. But if one looks at the way people deal with numbers and information, one finds that longer word lengths are of limited use, particularly from the human factors point of view. Certainly a sixty-four-bit machine or a seventy-two-bit machine or a hundred-twenty-eight-bit machine is a useful device. The Star series, or whatever superlarge machines are used in scientific calculation—they're all sixty-four-bit or larger. Still, there's no doubt this road, too, is reaching the bit of no return.

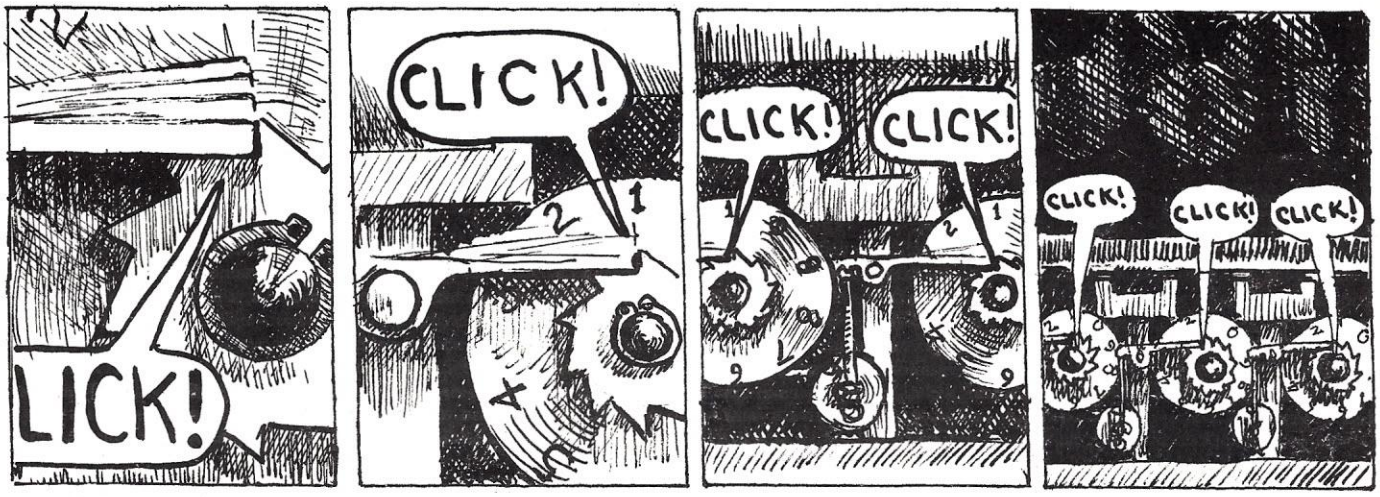
An approach which turns out to be more practical and useful is to deal with information in matrices. Which is why APL, the much-promised, never-quite-arrived home computer language, which requires a special instruction set and funny keyboards which no one will get for you, but everybody will promise there's one for two hundred dollars somewhere, may well be the personal computer language of tomorrow. It allows you to handle information as a matrix.

Let's consider a two-dimensional matrix. (A matrix is just some number times some other number. For instance, a crossword puzzle with 128 vertical squares and 128 horizontal squares, or the Dazzler in high resolution mode, which happens to be the same matrix.) When you begin to process things as matrices, a lot of very difficult mathematics and comparisons become simplified. Instead of shoveling a lot of words into the computer, trying to fit them into memory as a matrix, and then bringing them back out as words, we have the whole matrix and we just compare matrix A with matrix B.

The problem is that our current programming logic doesn't fit very well into matrix programming. Take "Jump if not zero" for instance. If you have a 128 by 128 by 1 deep matrix, you have 2K of computer core. There are lots more possible conditions in there than just "Jump if the whole thing exactly matches or not." You begin to get statements that say, "Go do something else if twenty percent of it, any twenty percent, compares with any other twenty percent." Strange things begin to happen. The possibilities for computer graphics in particular are amazing. And I think it's only with matrix processing that computer graphics will come into its own.

If you have matrix processing, then instead of looking at a Dazzler picture as a whole set of words put together, you will look at a Dazzler picture as *one* computer word which has height and depth. If you get 512 by 512 by 1 matrix processing, you can handle full-resolution video pictures as single computer words.

Even greater possibilities arise when you consider that matrices don't have to be limited to two dimensions—they can be three dimensional. They can be, say, 512 by 512 by 8 bits deep. An eight-bit processor at every point in the matrix. This would be extremely expensive now. But like all computer technology, if you went out and made a


LICK!
CLICK!
CLICK!
CLICK!
CLICK!
CLICK!
CLICK!

BABBAGE AND LOVELACE

Cartoon by Steve Gerling

million of them. . . . Then we'll have even more of a software gap. For programming will have to undergo certain perceptual as well as conceptual changes to deal with matrices.

Instead of absolute conditions, matrices lend themselves to percentage comparisons. You can still AND, OR, EXCLUSIVE OR, ADD, and SUBTRACT matrices, MULTIPLY and DIVIDE matrices, but you're also going to need computer-type commands that start to do things if a certain *percentage* of something compares to another certain *percentage* of something else. The whole logic then begins to head towards what we think of, heaven forbid, as analog thinking.

If you look at languages which deal with matrices, Ken Knowlton's EXPLORE language for one, you can do things like define a certain area within a matrix and then perform a logical comparison within that area of the matrix. It's similar to masking bits on a single computer word. But you're going to have to get into a whole new area of logic. What do you get in return? Speed. Lots of speed.

Exactly what's going to happen remains unknown. But the fact is that it's relatively easy to build matrix machines, or it's becoming more and more reasonable to do so. And we're running out of directions to build non-matrix machines. The limited gains in data processing produced by word length increases become even more obvious when you consider that even by going as far as a 100,000-bit long word length on a computer, we couldn't get anywhere near as far as we would with a 512 by 512 matrix, in terms of real power. This is where things will change.

Now the part about the change that's hard, as I've said, is not the hardware. IBM could obviously take something similar to the 8080 and put 512 by 512 of them on a chip and make a couple of million units and reduce the cost enough to get everyone interested. In fact, IBM has built special-use matrix processing machines for the government. There are a couple that are used for actual graphic comparisons. You take a picture, or, more to the point, a map, and it goes through as one thing and you compare it with the next. What the maps are all about, and in which crucial function this matrix processing serves, I'll leave up to your imagination and to the government. The point is that on a limited and budgetless basis at least, matrix processing is already here.

The real problem comes in the software. Our present computer software is in a period of stabilization. It all appears somewhat similar. If you know machine language for the 8080, you can understand machine language for the PDP-11. If you know machine language for the 8080, BASIC is no problem, FORTRAN is no problem, even LISP and APL and TRAC are no problem. They all make sense relative to FORTRAN, relative to machine language. You're doing the same things with loops. Either they're interpreted or they're compiled, and there are very few other things that can be done with them.

With matrix processing, the whole ball game is changed. A certain number of people are not only going to be left behind, whole cadres of programmers are going to be left totally in the dark. The same mind that's required to do a very concise "Jump if zero, jump if not zero," is rarely the type of mind that can deal with "Jump if forty percent looks like. . . ." And those are precisely the sorts of things programming will get into.

How do you do a "Jump if forty percent looks like" on the machine level? Well, you set up a matrix command structure that does the logical OR over the whole thing and then counts percentages of parts of the matrix by looking at it. And this whole change in architecture will probably come around pretty rapidly. It's not that different from the hypercube, which was what IMSAI got a lot of power out of suggesting and which was parallel processing with lots of 8080s. But parallel processing is of much more limited use than building a machine that works on the whole matrix, with all the memory planes going together.

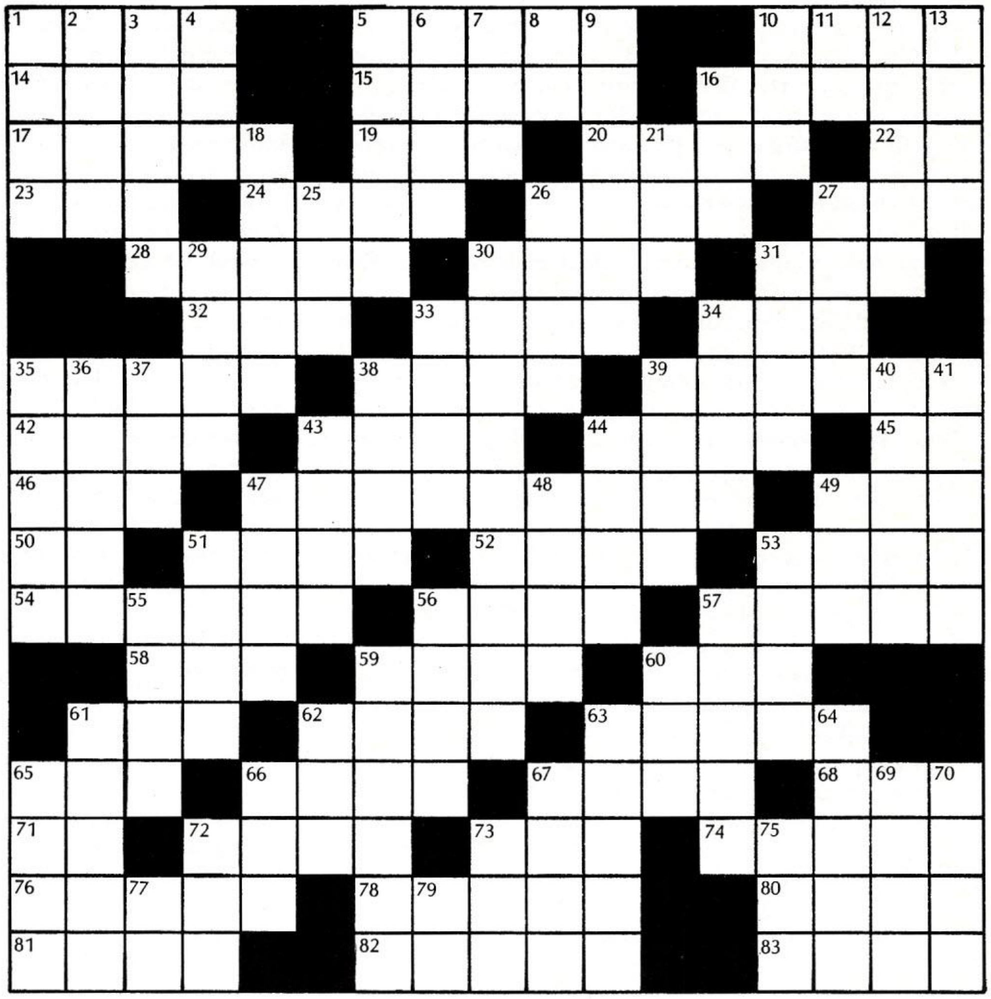
What matrix processing is, is a better pipeline. Now we take all the information through on one single plane. With matrix processing, instead of passing everything through a long narrow slit, we've got this huge square or cubical hole we shove it down through. And we can process information a lot more quickly. But you have to think about what the information is or how it's useful in a different way. It's very unlikely that you'll get two 512 matrices that are exactly the same. But you'd get parts of them that are the same. This will require a form of logical comparison that will change the absolute way people usually think about computer programming—once matrix processing takes hold, it's going to be a matter of playing the percentages. ▼

CLICK
CLICK
CLICK
CLICK
CLICK
CLICK
CLICK
CLICK
CLICK
CLICK
CLICK
CLICK

"My goodness, Babbage, what on earth is it?" *"Ripple carry."*

PROMpuzzle

by Daniel Alber



The solution to this PROMpuzzle will appear in next month's *ROM*.

ACROSS

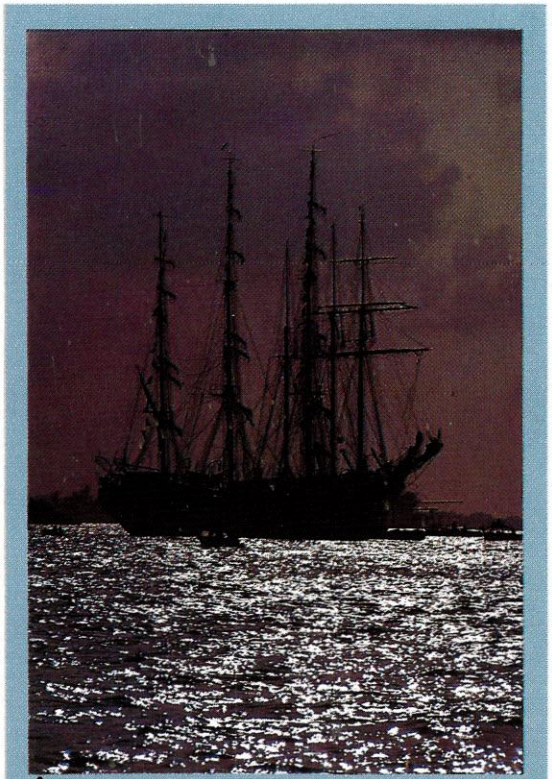
1. Process for locating specific computer data
5. Type of cheese
10. Part of subprogram that connects with main program
14. Bank abbreviations
15. Belief
16. Gay ____
17. Decorate
19. Total up
20. Highway
22. ____ the people
23. Weekday (abbr.)
24. Broadcasts
26. Transmitted data
27. Lady bird
28. Displace an ordered set of computer characters
30. American beauty
31. Knight's title
32. Capone or Capp
33. Places a binary cell in the 1 state
34. 2,000 pounds
35. Maple genus (plural)
38. Impudence
39. Placing data in storage at a specified address
42. IC
43. Mata ____
44. Deposition shield
45. Venus ____ Milo
46. Abode (abbr.)
47. Some digital-computer devices
49. Zodiac sign
50. Letter
51. Food fish
52. Prefix used with meter or printer
53. Ku Klux____
54. ____ transformer
56. Habit
57. Fish organs
58. Slice
59. Planet
60. "For Me and My____ "
61. Graduate degree (abbr.)
62. Takes notes
63. Floating gate Avalanche-injection MOS
65. Bovine sound
66. Crazes
67. 4096 consecutive bytes
68. High speed memory (abbr.)
71. Two (Rom. num.)
72. Hollywood jungle boy
73. Hawaiian dish
74. The theoretical basis of a numerical system
76. Give a dirty look
78. Optical maser
80. Persia
81. "____ Wars"
82. Some computer commands
83. Party members (abbr.)

DOWN

1. Thailand
2. Within (comb. form)
3. Short jackets
4. Keyboard send/receive
5. ____stop multivibrator
6. Marries
7. Hoosier State (abbr.)
8. Compass reading
9. Computer language for structural-analysis problems
10. Boy
11. Information retrieval (abbr.)
12. More recent
13. Sharp
16. Butter slice
18. Brads
21. ____cell switching
25. ____, ands or buts
26. Drunkards
27. Rear
29. Stringed instrument
30. Electricity flow restrictors
31. Wet
33. Indian dress
34. Mrs. Dick Tracy
35. Land measures
36. Treasure ____
37. End interruption sequence (abbr.)
38. Wise man
39. Unusual
40. ____ noise diode
41. Some lights
43. Assist
44. Thaw
47. Retreat
48. Decades
49. Low-level logic
51. Potato (sl.)
53. Prefix with watt or cycle
55. ____ check
56. Wide area telephone service (abbr.)
57. Monopoly or chess
59. Discrete program
60. Choke
61. ____junction transistor
62. Poke
63. Bazaars
64. Both use computer time
65. Hit or____
66. Distant
67. Bard
69. ____ package
70. Not women's
72. Lecture ____ (abbr.)
73. Greek letter
75. El ____
77. Apiece (abbr.)
79. Commercial

A magnificent first edition . . . pictorial memories of a moment America will never forget . . . or see again

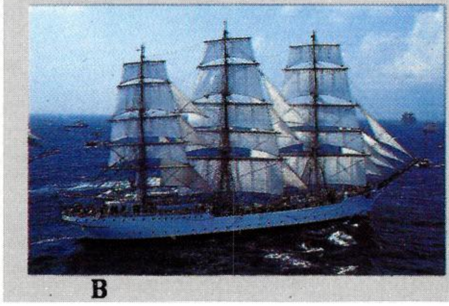
THE SAILING SHIPS


A

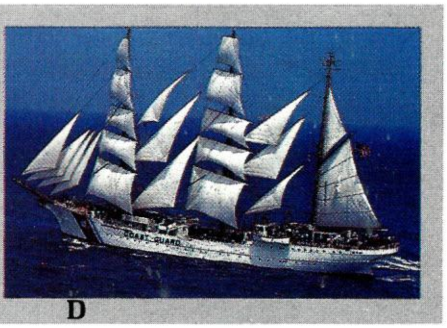
Professional quality lithographs of the majestic Bicentennial Ships, now available to the public in an exclusive, limited edition from America's foremost engraver-printer, COLLIER GRAPHICS.

handsomely matted
in silvery metal frames; $25 each

unframed, $10 each


B

C


D

THIS IS A LIMITED EDITION . . . ORDER TODAY WHILE THEY LAST

Never before offered to the public, these magnificent 12x19 full color lithographs of the tall-masted SAILING SHIPS were originally photographed for professional use only!

Now, you can own and enjoy their historic beauty, their incredible clarity, color and reproduction, which gives them almost a three-dimensional quality. And they are only available from COLLIER GRAPHICS — who provide the superb color graphics for America's leading advertising agencies and publishers.

Perfect for your home, boat or office. A lasting gift. Order a set for yourself and one for your children . . . to keep always.

COLLIER GRAPHICS INC., 240 West 40th Street, New York, New York 10018

Please rush the following SAILING SHIPS, securely packaged and postage prepaid, with the understanding that my purchase is UNCONDITIONALLY GUARANTEED by you if the order is returned within 15 days of delivery:

Send me the following print(s). Quantity ________ Price ________ Total ________

A. Christian Radich ________

B. Danmark ________

C. Kruzenshtern ________

D. Eagle ________

Please add $2.50 for shipping and handling for framed print(s) or $1.00 for unframed print(s).
(N.Y. State residents add applicable tax, NYC residents add 8%. Allow 6 weeks for delivery.)

Name ________

Address ________

City ________ State ________ Zip ________

Enclosed, check or money order for $________
Or charge my credit card ________ Master Charge ________ BankAmericard

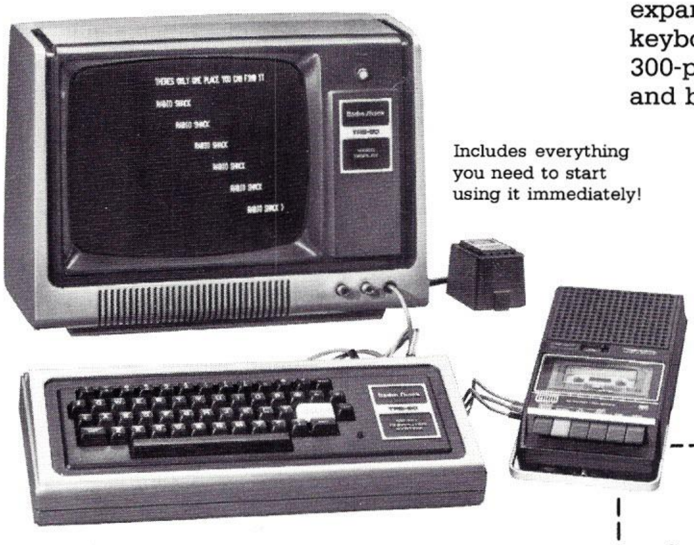
Credit Card # ________ Inter Bank # ________ Expiration Date ________

Signature X ________

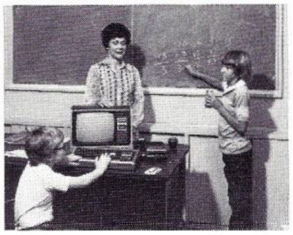
The first complete low-cost microcomputer system for home, business or education!

Radio Shack TRS-80

The TRS-80 is for people who want to use a computer now—without the delay, work and problems of building one. The system is fully wired, tested and U.L. listed—ready for you to plug in and use! Program it to handle your personal finances, small business accounting, teaching functions, kitchen computations, innumerable games—and use Radio Shack's expanding line of prepared programs on cassettes. The Z80-based system comes with 4K read/write memory and Radio Shack Level-I BASIC stored in read-only memory. Memory expandable to 62K bytes. Includes CPU, memory, keyboard, display, power supply, cassette data recorder, 300-page manual, 2-game cassette program. Designed and built in USA by Radio Shack. Only 599.95.



Includes everything
you need to start
using it immediately!



Clip and Mail Coupon Today!

C012

Mail to: Radio Shack, Dept. TRS-80
205 N.W. 7th St., Ft. Worth, TX 76101

Send me more data on the TRS-80 microcomputer

- **Description of applications, software and peripherals available through Radio Shack • Owners' newsletter**
- **Price list • List of stocking stores and dealers**

NAME ______________________ APT. NO. ______

ADDRESS ______________________

CITY ______________ STATE ________ ZIP ________

SOLD ONLY WHERE YOU SEE THIS SIGN:

Radio Shack®

A TANDY COMPANY • FORT WORTH, TEXAS 76102
OVER 6000 LOCATIONS IN NINE COUNTRIES

Price may vary at individual stores and dealers